DIPLOMA THESIS

Development of the CAN-part of a redundant elevator control system

Submitted at the Faculty of Electrical Engineering and Information Technology, Vienna University of Technology
in partial fulfillment of the requirements for the degree of
Diplom-Ingenieur (equals Master of Sciences)

under supervision of

Dietmar Dietrich, O.Univ.Prof. Dipl.-Ing. Dr.techn.
Dietmar Bruckner, Univ.Ass. Dipl.-Ing. Dr.techn.

by

Tarik Kurtovic
Matr.Nr. 0227061
Argentinierstrasse 28, 1040 Vienna

Date _____

Kurzfassung

Die Aufnahme von Diversität in den Entwurfsprozess mit dem Ziel der erhöhten Zuverlässigkeit ist bereits eingesetzt worden. Als Ziele der erhöhten Zuverlässigkeit eines Systems zählen quantitativ messbare Gründe wie eine erhöhte Sicherheit, sowie qualitative Merkmale wie der erhöhte Benutzerkomfort. Diversität der Hardware wird im Entwurf der in dieser Diplomarbeit entworfenen Aufzugsteuerung durch die Verwendung von unabhängigen Mikrocontrollern und Kommunikationsnetzen des jeweiligen Steuersystems berücksichtigt. Diversität der Software ist durch die für jedes Steuersystem unabhängig voneinander entwickelten Algorithmen zur Auswertung der Sensorwerte, Positionsberechnung und Kommunikation gegeben. Die praktische Aufgabe ist die Entwicklung einer Aufzugsteuerung unter Verwendung von vorhandenen Sensoren und Entwicklungsplatinen. CAN findet Anwendung für die untersten zwei Schichten des ISO/OSI Referenzmodells, auf die eine für den verwendeten Mikrocontroller portierte Open-Source-Realisierung des Protokolls CANopen setzt. Es ist die praktische Implementierung einer Aufzugsteuerung dargestellt, die auf einem Universalmikrocontroller sowie auf Open-Source-Realisierungen des Kommunikationssystems und Betriebssystems basiert. Der Entwurf baut auf einem Top-down-Ansatz auf, mit Hilfe dessen von den allgemeinen Systemanforderungen und dem Sicherheitsziel erst die funktionale und die Sicherheitsanforderungen und weiter die Anforderungen an die Hardware und Software abgeleitet sind. Die auf einem Schichtenmodell basierende Architektur dient zur Darstellung und Aufteilung der Software in wiederverwendbare Module. Die in dieser Diplomarbeit dargestellten Konzepte können in anderen Steuersystemen zur erhöhten Zuverlässigkeit und Fehlererkennungsfähigkeit eingesetzt werden.

Abstract

The application of diversity in the design process has been used for some time with the aim of increasing reliability of a system. An increase in reliability is advantageous on objective grounds such as increased safety, as well as for subjective reasons such as improved user comfort. Diversity is taken into account while designing an elevator control system with hardware diversity present in the form of independent microcontrollers and communication systems used to build each of the two control systems. Software diversity originates from independently developed algorithms for sensor value processing, position calculation and communication within each system. The practical task is that of developing an elevator control system using the available sensors for input and evaluation boards as hardware platforms. CAN is used for the first two ISO layers in this part of the complete diverse system, with an open source implementation of CANopen, ported for the microcontroller used, as the higher layers of the communication model. A practical implementation of an elevator control system based on general-purpose microcontrollers and open source communication stack and operating system is shown. A top-down methodology is used to arrive from general requirements and the safety goal, over safety and functional requirements, to hardware and software requirements that can be mapped onto design artifacts. To enable a clear split of software into reusable modules a layered software architecture is adopted. The pre-sented principles can be used in any control system to ensure improved reliability and error detection capabilities.

Acknowledgements

I would like to express my gratitude to my supervisors

O.Univ.Prof. Dipl.-Ing. Dr.techn. Dietmar Dietrich
and Univ.Ass. Dipl.-Ing. Dr.techn. Dietmar Bruckner

for their help and guidance during my master studies and the work on this thesis.

Thanks to my parents for making so many things possible and influencing me in ways I still keep discovering, and to my sister for always being there.
I would also like to thank my wife for her support when motivation was not in ample amount.

Table of Contents

Abbreviations

ADC	Analog-Digital Converter
ARM	Advanced RISC Machine
AUTOSAR	AUTomotive Open System ARchitecture
CAN	Controller Area Network
CiA	CAN in Automation
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
DAC	Digital-Analog Converter
ECU	Electronic Control Unit
EXTI	External Interrupt
GNU	GNU is Not Unix
GPIO	General-Purpose Input/Output
GPT	General-Purpose Timer
ID	Identifier
IDE	Integrated Development Environment
IP	Intellectual Property
ISO	International Standardization Organization
LED	Light-Emitting Diode
LIN	Local Interconnect Network
LON	Local Operating Network
MCU	Microcontroller Unit
NMT	Network Management
OS	Operating System
PDO	Packet Data Object
PLL	Phase-Locked Loop
PWM	Pulse-Width Modulation
RCC	Reset and Clock Control
RDO	Reception Data Object
RISC	Reduced Instruction Set Computer
RT	Real-Time
SCI	Serial Communications Interface
SDO	Service Data Object
TIM	Timer
TDO	Transmission Data Object
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very High-Speed Integrated Circuit
XINU	eXellent Interface for Non-haptic Use

1 Introduction

Various kinds of elevators and elevator control systems have been proposed and used throughout the years. Arguably one of the first designs employing safety elements was the elevator introduced by Elisha Graves Otis in 1854 which featured an automatic braking system that was actuated automatically in the case of a cut rope [1]. Throughout the decades, other possibilities for and areas of improvement have emerged. Some of the aspects that can be addressed by using the latest advancements in the field of building automation are improving reliability and raising availability of time-critical systems. Apart from these goals, advancements in electronic control systems have given rise to possibilities of reaching more complex goals. Some of these goals have been researched in [CLXC06], such as the minimization of the average wait and ride times of passengers, the percentage of passengers waiting more than 60 seconds for an elevator car to arrive, the overcrowding rate and the power consumption of an elevator group. The main point behind this breakdown into separate criteria is that it enables the definition of concrete goals which, in turn, give clues on the attainment of non-functional qualities of a system such as those of a system having a "high availability". In this way, a normally abstract and hardly testable non-functional requirement can be covered by one or several concrete criteria that provide information on a specific non-functional requirement. The introduction of such criteria also enables possibilities of classification and thus easier comparison of similar implementations.

The concrete goals of lowering out-of-operation times and adding of diagnostic functions are facilitated by use of electronic control systems as well, and they actually work hand in hand. Similar to using the diagnostic equipment in a car repair shop or representatives office, maintenance personnel can be provided with current information on the state and health of the elevator control system. Such additional functions would then serve to increase the reliability of an elevator system by using diagnostics to detect whether a certain module is about to fail and being able to replace it before it does. Even though this would not directly improve the availability since the system, or parts of it, would still have to be put out of operation while the repair is ongoing, there is one major advantage to using diagnostics. This advantage is to be seen in the ability to plan maintenance sessions so as not to interfere with times of peak load and high demand, or high criticality.

1.1 Safety analysis

Although the goals of raising reliability and availability are important, they are not the only ones to be considered in a safety-relevant control system such as an elevator control system. In the

nomenclature of the ISO 13849 standard on the Safety of machinery [Eur07], it is implied that some parts of a control system are charged with a certain safety function, while others are not. Further, safety-relevant parts of control systems are defined as such hardware and software parts that provide a certain safety function. The safety function is in turn defined as the function of a machine, failure of which can lead to an immediate increase of the risk (or risks). A safety-related part is one that reacts on safety-related inputs, and generates safety-related outputs. Figure 1.1 illustrates this relation between the three concepts.

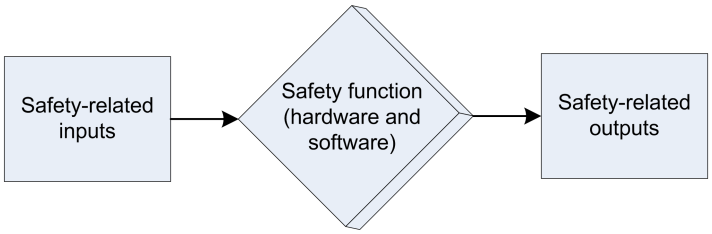

Figure 1.1: ISO 13849 view of a safety function

On the example of an elevator control system, the condition that an elevator car must not be allowed to move while its doors are open can be considered as a safety function. The absence, i.e. failure, of this safety function leads to an elevated risk since the elevator car would be free to move exposing persons and/or material goods to danger of injury or destruction of goods. The safety function itself would be spread out over various software and hardware, with inputs from sensors for elevator car and door positions, actuators that open the doors and lock the elevator car in position, as well as algorithms that calculate the outputs to the actuators from the input information from the sensors. The safety function could be distributed over various system parts, not necessarily being encapsulated in a single hardware component with the corresponding safety-related software running on it.

One of the approaches to guarantee the maintenance of the safety function is to replicate the safety-related parts themselves. This approach is illustrated in Figure 1.2.

The premise of the approach of triple modular redundancy is that each of the three units produces the same output (decision) if given the same input. The temporal behavior, such as with which granularity the units present the outputs, is not explicitly given and is currently out of scope. The decision of each unit is typically a binary value. The voter presents the verdict of a majority-vote at the output and produces correct results as long as two modules provide it with correct results [MSM99]. An implicit premise of this approach is that the units should be independent from each other once in use, apart from sharing the same design. This premise is also a latent problem of the approach of triple modular redundancy, since it does not cover common-mode failures, such failures which affect more that unit at the same time. Common-mode failures can be a result of an error in the design of a unit, e.g. an error of the logic under specific circumstances; but they can also result from external sources, such as disturbances in a shared power supply or shifts in a shared ADC reference voltage.

One method used against common-mode failures, also selected for this project, is that of diversity in hardware and software. As the results of simulations in [MSM99] illustrate, diversity is a very effective way to combat common-mode failures and design faults, and is superior to simple multiplication of the same unit. However, the effects, or gains, seem to be dependent on the mission time. The best results are tolerance against common-mode failures is observable when

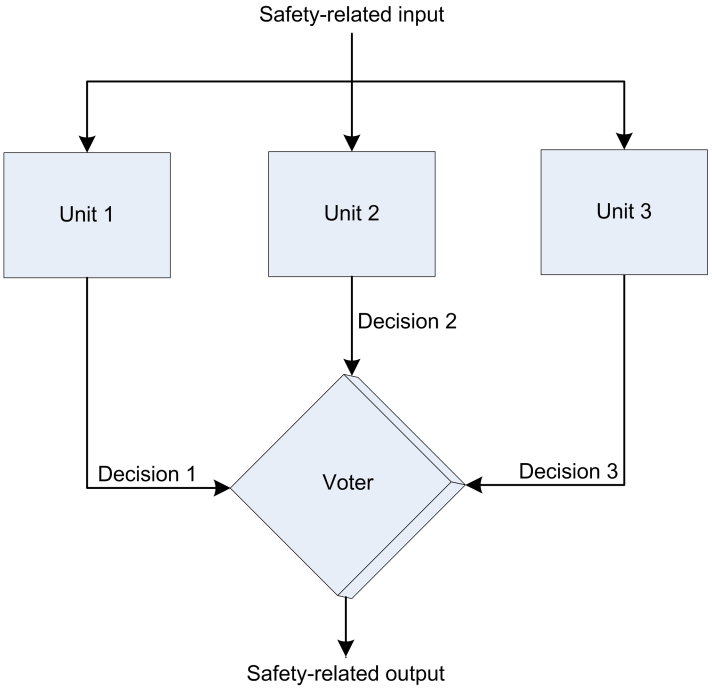

Figure 1.2: The traditional approach of triple modular redundancy

the mission time is in the range of under 10% of the Mean Time to Failure of one unit used in simplex mode (without a diverse unit and a voter).

Similarly, in presence of multiple independent failures, diversity does not promise significant gain in reliability. The approach is, thus, that of an exhaustive analysis of the reliability for each system being designed.

1.2 System view

This master's thesis is part of a project designed to improve reliability of an elevator control system by using redundancy in the form of a combination of two control systems running in parallel, featuring hardware and software diversity. The diversity is present on each level of design, from sensors using different concepts to obtain the measurement values, over different network structures and protocols, to different software running on different hardware platforms. The way the two diverse control systems are to communicate over a common serial line, using a protocol for the exchange of data between the two systems as well as the access to the communication line.

A safety concept is defined as well, according to which the control system is developed, including the identification of the risks, the safety goal i.e. the safe state, as well as the safety function which serves to contain the risks. The safety function is analyzed and hardware and software mechanisms that are to be used to provide it are outlined.

The system can be seen as a three-plane structure presented Figure 1.3 in with the bottommost plane consists of sensors, each of which is connected to both of the buses (control systems). The middle plane is that of the controller boards with network access hardware and software, and

the application running on one central controller or distributed over several controllers of that network. Another part of the middle plane is the resolution controller, which has the task of error resolution and marshaling of control between the two systems.

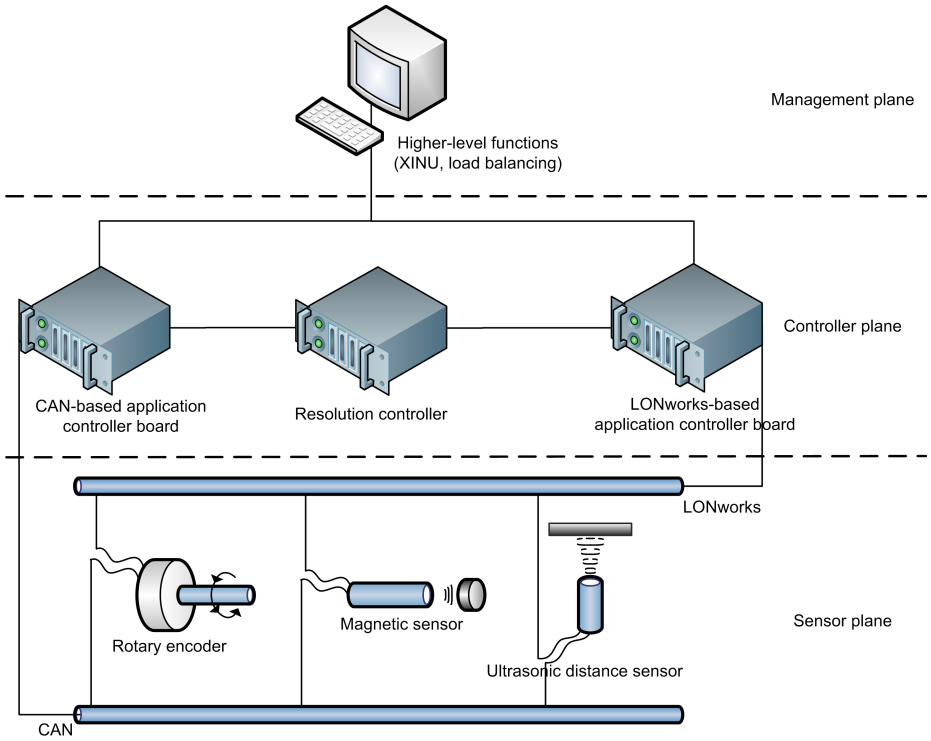

Figure 1.3: Three-layer structure of the entire elevator control system

The top, or management, plane has the task of connecting the middle - controller - plane with higher-level functions. The management plane could consist of a program running on a single computer, possibly supporting some more advanced features such as load balancing or wait and ride time minimization. The functionality of XINU is, in this representation, a part of the management plane.

The elevator control systems are to be connected via a resolution controller that is out of scope of this thesis. Its aim would be to analyze data coming from the application boards of each control system and reach a decision which of the two control systems should be in charge of controlling the elevator motor, and thus its functionality.

The controller plane, originally viewed as a single controller, can be configured redundantly or in a distributed manner. The advantages of these approaches shall be outlined and weighted; the outcome influencing the topography to be implemented. The thesis tackles in detail the application of the architecture and elements of Controller Area Network (CAN) in the field of building automation through its use as one of the communication buses in the redundant control system. Existing applications of CAN in the field of elevator control have been researched, yielding possible improvements in the sense of interoperability and design concepts proven in use.

1.3 eXellent Interface for Non-haptic Use

Main research goals of the XINU (eXellent Interface for Non-haptic Use) project are assesment and selection of existing methods feasible for non-haptic control. The project task of a design and evaluation of specific command sets is matched by a flexible architecture build around a camera input system to provide a practical implementation of a non-haptic control that can discern between the users' head movements and control several devices in the field of building automation. The prototype is subjected to a phase of testing at an inclusive polytechnic high school in Austria [RPZH10].

The interface, open to all forms of communication as input, uses visual communication for the initial approach. A reduced set of distinctive movements, in the practical example head movements of the user which are recognized as data, is fed into systems for head pose estimation and face tracking. The change of head pose is recorded and mapped onto a set of commands. One of the disadvantages of using such visual communication as input is the need for previously agreed-upon commands that the user needs to memorize and reproduce in order to control the system [RPZH10]. The initial training phase was perceived and recognized as reasonable in the follow-up research paper [ZRP11].

In the context of the elevator control system presented here, the functionality provided by the XINU environment can be visualized in the management layer, as presented in Figure 1.3. XINU would provide input to the control algorithm in the form of commands further used by the control algorithm in acquiring the next set-point. Initially, an RS-232 connection shall be used as the physical interface to XINU.

2 State-of-the-art in automotive communication systems

To aid in the knowledge necessary to develop a system for elevator control the state of the art and current developments in the field are considered. A description of three communication systems widely used in the automotive industry is given. The presented communication systems are compared according to criteria that is important when choosing a bus system for elevator control applications. Finally, one bus system is selected to be used in further development.

One of the basic goals of this thesis is the evaluation of several and selection of one communication protocol to be used as the communication backbone in the implementation of one part of a redundant elevator control system. This chapter shall present some of the communication buses currently used in the automotive field, with a focus on their properties that could be useful in designing an elevator control system with an accompanying comparison and selection description.

2.1 Overview

Several protocols with properties interesting to the area of building automation have been in use in the automotive industry. Each of these satisfies a certain set of requirements, some of which overlap with each other and most of which overlap with those for a communication system usable in the area of building automation. A single communication bus shall be selected alongside the criteria that influenced the selection process. Communication methods and protocols used in the automotive industry are numerous, some of which are aimed toward very specific fields of application. An analysis of a large number of such communication systems is beyond the scope of this thesis and this is the reason why further systems, such as MOST, are not dealt with in detail.

A major criterion for selection of a protocol used in the automotive industry is the breadth of use and extent of tests carried out on such systems, mostly connected to the sheer number of units produced which incorporate them. One might argue that a communication protocol used in the aerospace industry would be even better suited to this argument and that would be true. However, controllers, transceivers and evaluation boards supporting protocols from the aerospace field are rare and expensive. On the other hand, communication components for automotive protocols are typically low in cost and available on evaluation and development boards from various manufacturers.

2.2 Controller Area Network

With the version 2.0A of the transfer layer published in 1991 and version 2.0B in 1995 the CAN protocol laid out by Robert Bosch GmbH has been in use in the automotive industry for a very long time. An effect of this long and extensive use is the wide availability of dedicated transceivers and controllers, with [RBF99] stating 11 million chips sold by the end of 1997. The large numbers of produced components in turn motivated a sink in the prices of components which, with the decision to offer the controller in form of VHDL code, helped broaden the availability of affordable microcontrollers with on-chip CAN controllers.

The ISO standard 11898-2 on the Controller Area Network, High-speed medium access unit [ISO03], stipulates the electrical parameters that a CAN transceiver is to implement. These include nominal and absolute ratings of bus and line voltages in the active and passive state, parameters of nodes disconnected from the CAN bus as well as the values of termination resistors to be used. Values for the bus length, with the recommended topology of a line structure, cable stub lengths and distances between neighboring nodes are also determined for the bit rate of 1 Mbit/s. These characteristic values are illustrated in Figure 2.1 The standard gives a hint that, at lower bit rates, the bus length may be lengthened significantly, as well as cable stub lengths and node distances, although without giving recommendations on appropriate values to be used for certain bit rates.

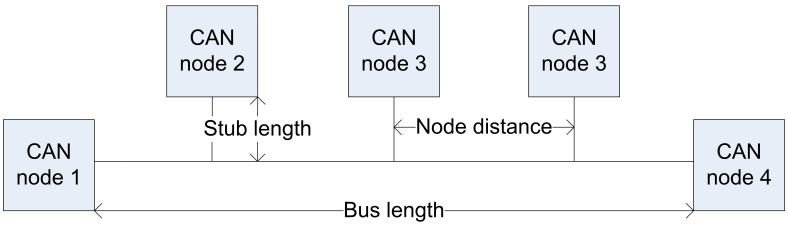

Figure 2.1: CAN bus as a line architecture with its basic parts

The basic term of communication in CAN is a frame which can be one of the following four types: data frame, remote frame, error frame or overload frame. Data frames are the most common form used for exchange of data, while remote frames can be used to request data in a polling manner. Error frames used by nodes in a CAN network to signalize whether they are in one, and in which, error state. An overload frame is a special kind of frame that is used by a node to exert back-pressure to senders, i.e. to request a delay of the next data or remote frame [Bos91]. Some of the basic concepts outlined in [Bos95] and commented for clarity are:

- Prioritization of messages: through arbitration of the message field containing the message Identifier and Remote Transmission Request bits;
- Guaranteed latency: caused by arbitration. Highest priority messages will have very low latency, but low priority messages can in theory be delayed indefinitely;
- Configuration flexibility: use of standardized connectors and a communication matrix typically enough to provide an integration interface;
- Multicast reception: each node in a CAN network can receive all messages on the network if desired, thus no need for point-to-point transmission that might cause higher network load;

- Error detection and signaling: CRC calculated and sent with each frame. Acknowledgment bit is set by each node receiving a message with a correct CRC sequence;
- Automatic retransmission of corrupted messages as soon as the bus is idle again: no reaction from microcontroller needed to retransmit a message thus possibly faster but potentially raises the short-term network load.

The specification, standardized by ISO under the ISO-11898 series of standards, foresees two modes of usage, that of standard and extended frames. The most significant difference between the two is the introduction of an enlarged identifier field in frames, being raised from 11 bits in the standard to 29 bits in the extended mode. Since the CAN specification versions 2.0A and 2.0B only cover to some extent the Physical and Data Link layer of the OSI Reference model [Bos95], as illustrated in Figure 2.2, a decision by the network and application designers is needed with regard to the potential implementation of upper layers.

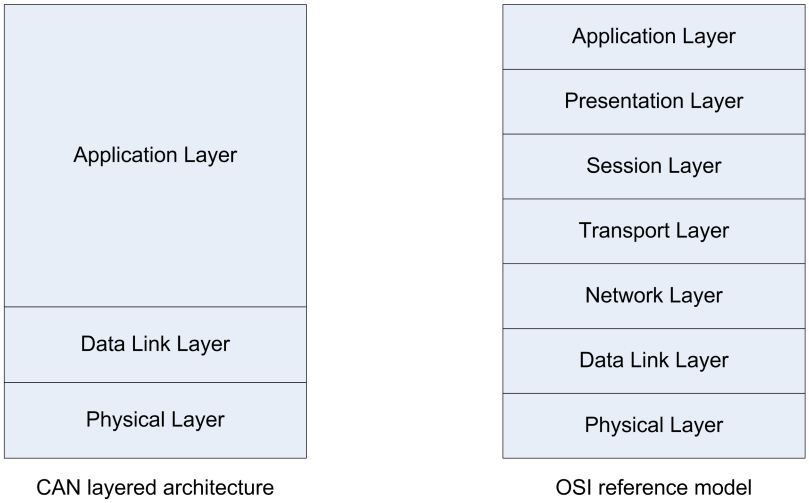

Figure 2.2: CAN layered architecture compared to the OSI reference model

This approach gives flexibility in designing the application, with the ability to optimize the run-time performance by eliminating unnecessary higher-layer functionality. However, it can cause confusion in practice since it is effectively left to the system designer to make decisions on a very level of abstraction, e.g. which kind of byte order to use.

The CAN specifications do not stipulate a transmission medium [Bos91] [Bos95], so as to allow usage of different potential transfer media. The ISO standard [ISO03] however specifies electrical properties of components used in building a high-speed CAN network with transmission rates of up to 1 Mbit/s. Network topology parameters such as bus and cable stub lengths, or rather their extremes, for a bit rate of 1 Mbit/s are also specified in [ISO03] with a hint that the values may be significantly different for lower bit rates. Practically, bus length depends on the bit rate since each node needs to be able to read an arbitration bit and possibly react on it, all in the period of one transmitted bit. The users and manufacturers group "CAN in Automation" outlines recommendations of transmission rates for industrial applications in the form of a draft standard with the designation CiA 102 [CAN10a]. From these transmission rates i.e. the related nominal bit times and the propagation delay of a copper twisted-pair cable, the maximum bus length can be calculated. A further factor is the length of stubs, i.e. cable distance from the main trunk of the communication bus which should generally be kept as short as possible [CAN07].

The CAN protocol uses an access scheme called CSMA/CD+AMP, for Carrier Sense Multiple Access Collision Detection and Arbitration on Message Priority. The carrier sense functionality ensures that a transceiver can only send data if the bus is idle, while the arbitration mechanism enforces a collision-free bus in the arbitration phase.

2.3 Local Interconnect Network

The Local Interconnect Network has been conceived and published by the LIN consortium in the form of a specification package in 1997. The main goal was to outline a low cost network for the automotive field, not only on the physical level specifying the integration and development process as well.

Some basic properties of LIN are outlined and commented [LIN10] :

- Single master with multiple slaves: request-response based protocol;
- Implementation reusing an existing UART/SCI unit or with dedicated hardware;
- Self synchronization in slave nodes: nodes synchronize on the sync field of messages send by the master node;
- Deterministic signal transmission and predictable behavior: static time-triggered communication initiated and governed by the master node;
- Speed up to 20 kbit/s: low speed useful for simple devices, non time-critical devices;
- Total length of bus line of 40 m: a significant constraint for building automation;
- Transport layer and support for diagnostics: concepts of a signal and a Packet Data Unit, as well as support for diagnostics on the transport layer.

Compared to CAN, the LIN specification defines in a single document the physical layer, the protocol layer which is a simplified implementation of an OSI Reference Model Data Link Layer and the Application Programming Interface comparable to the Application Layer.

The only conceptual difference of the software running on the master node to that running on any slave node is the addition of a master task which sends out a communication request, called a header in LIN terminology, and to which the slaves send responses. The logic of communication is such that a master task has to broadcast a request also in the case of sending data from the slave task of the same (master) node. Since all communication is broadcast over the same one-wire interface, each slave listens to the communication and can act upon the data of interest it receives. There is no way for slaves to send messages on their own, i.e. without request coming from a communication master. This represents a single point of failure in such a network, for if the master stops sending communication requests the communication is paralyzed.

The LIN network specification is very appealing to evaluation from the point of view of availability of low-cost hardware and standardized interfaces. However, the limitations in form of a low maximal transmission rate of 20 kbit/s and a total bus length of up to 40 m are too narrow for the field of building automation.

2.4 FlexRay

FlexRay is a time-triggered communication protocol specified by the FlexRay Consortium, an organization that specified the use of its IP (intellectual property) to automotive purposes. Apart from the protocol specification, the physical layer has been specified in terms of general physical layer parameters that have to be fulfilled. Thus it is possible to have a FlexRay network using a twisted pair copper-wire as the physical medium, or one using an optical fiber. The main requirement is that the medium can represent a binary signal, and fulfill the timings given in form of physical layer parameters, outlined in [Fle05].

Differently to CAN and LIN, FlexRay is time-triggered at its core. While a similar statement could be made with regard to LIN with some constraints, in the sense that the communication can be defined according to a fixed schedule causing the master node to send requests at fixed periods, this is not the main idea behind the LIN protocol.

With FlexRay, however, the notion of time-triggered communication permeates all aspects of the protocol. Indeed, three main terms used throughout the FlexRay Protocol Specification [Fle05] are those of "cycle time", "macrotick" and "microtick". Cycle time is the time within the current communication cycle. A macrotick is used to denote the smallest granularity unit of the global time, derived using a algorithm that synchronizes the time within the complete FlexRay cluster, while a microtick is a duration derived from the nodes oscillator and is not affected by the synchronization algorithm. Microticks are used to adjust the length of the macrotick of a node to the macrotick duration calculated at this node by the algorithm fault-tolerant algorithm. The protocol offers a baud rate of 10 Mbit/s which, taking into account the overhead caused by the header and trailer segments (of 64 bits), yields a combined maximal throughput of 7.987 Mbit/s.

Each communication cycle is consists of static and dynamic segment, with the outline of the cycle being defined using a static schedule. Thus each node that is supposed to send information has a part of the static and/or dynamic segment reserved. This is one of the trade-offs of FlexRay, i.e. that the practical network throughput in case not all nodes need to send data is far below the theoretical maximum throughput.

The advantage of the compromise is the global time-base which makes response times significantly shorter than with CAN, since it is known before-hand when data from a node should arrive. This also means that the timeout value of a frame (the time within a specific frame has to arrive) signifying an error can also be set to a shorter duration.

Parts of the static segment are assigned to various nodes by the network designer are part of a cluster-wide communication schedule definition. Parts of the dynamic segment of each application cycle can be reserved for nodes that might have additional data to send. If a node has information to send in the dynamic cycle it sends a frame; otherwise all nodes can detect that there was no activity within the macrotick and the new slot of the dynamic begins.

One disadvantage of using FlexRay in any application comes from one of its basic properties: the static schedule that is the basis for communication. At the network design stage of system design, a network designer would outline the schedule using a schedule authoring tool, assigning slots of the static and/or dynamic segment to various nodes of the network. In the ideal case of an omniscient network designer, the schedule would only have to be generated once for the complete system and loaded into each node. In real world applications the issue becomes largely one of logistics and propagating changes in network configuration. It is much more critical for

each node in a FlexRay network to have knowledge of timing of slots assigned to itself, as well as to the nodes it is to receive information from, than that is the case in CAN or LIN.

The solution to this problem requires use of further tools for propagating changes in the schedule and generating appropriate code for each node. The problem tends to multiply with the question of interoperability of nodes from various manufacturers using different platforms - thus surely needing generator tools adapted to their platform.

The problem, and its solution, have the potential to overwhelm non-experts in the field of FlexRay and do not seem very well suited for experimentation.

Similarly to CAN and in contrast to LIN, the FlexRay Protocol Specification does not enforce rules on the content of frames. Frames are presented as content carriers; how the information is structured within a frame is not within scope of the specification. There has been some standardization in the area of segmentation, e.g. the FlexRay Transport Layer specification by the AUTOSAR organization.

A complication that would arise from using FlexRay in the setting of a building automation system is the insecure legal situation. The protocol specification [Fle05] itself explicitly states that the protocol was neither developed nor tested for use in non-automotive applications. Although this statement does not imply a ban on use of this system as the communications backbone of a building automation system, it is unclear what the implications on the safety analysis and system acceptance would be.

A possibly more serious issue is that of high costs due to the additional equipment necessary for the layout of a typical network in building automation. In the case of a long copper-wire pair an active star, a kind of kind of bidirectional repeater, would have to be used every ca. 150 m. According to [Fle06b], the maximal number of active stars in a FlexRay network two which would imply a maximal bus length of around 450 m. However, [Fle06a] recommends maximal distance between two active stars of 24 m and the same value for the maximal distance between an active star and an ECU - bringing the total length down to only 72 m.

Another issue would be the computational power required by each nodes CPU required to implement the software side of the communication protocol, as well as the additional power that would be needed to run the CPU. With the trend of deeply embedded, low-power smart sensors/actuators a communications system it is not acceptable that the communication part of the device requires constant power and attention by the CPU.

2.5 Communication bus comparison and selection

Each of the three presented communication systems in use in the automotive industry have their advantages and disadvantages, the comparison of which is presented in Table 2.1.

The three compared protocols vary greatly with regards to the transmission speeds that can be achieved using each of them. LIN features the slowest bit rate of the three and is limited to 20 kbit/s. A single 32-bit value, such as could be given by a position sensor, would use 16% of the available bandwidth. Taking into account the possible need to transfer further measurement values, both simple (few bits) to more complex (e.g. 32-bit floating point) this limitation is prohibitive.

A further important criterion to use to select the bus system for an elevator control system is the maximum achievable bus length at the operating bit rate. The bus length depends on numerous factors, among which are the target transmission speed as well as the signaling mechanism used. The best choice in terms of bus length at maximum speed would be FlexRay as the transmission speed of 10 Mbit/s is feasible with a bus length of up to 72 m. CAN and LIN seem to achieve the same result, however the bit rate of a CAN bus at the bus length of 40 m is 50 times that of LIN using the same bus length.

Neither the CAN nor the LIN specifications make attempts to introduce the concept of timing for upper layers of a communication stack, while FlexRay includes such functionality.

With CAN it is the message with the highest priority, i.e. the lowest ID number, that 'wins' the arbitration and is broadcast on the bus. As a consequence, messages with ID numbers higher that the lowest one occurring in the system can be effectively barred from being sent on the bus in the case when the message with the highest priority is permanently being transmitted.

In LIN the master initiates communication with the nodes by sending requests. It would be possible to configure a LIN network in such a way that the master adheres to a schedule with a fixed sequence and timing, thus in effect creating a time-triggered communication basis.

FlexRay features time-triggering as the very communication principle. No communication master is present, but a number of start-nodes are required to initialize the network on start-up and to support the fault-tolerant algorithm for clock synchronization.

	CAN	LIN	FlexRay
Transmission speeds	1 kbit/s - 1 MBit/s	1 kbit/s - 20 kbit/s	2.5 Mbit/s - 10 Mbit/s
Bus length at maximum speed	30 m	40 m	72 m
Sense of global time	No	If master provides functionality	Yes, in all nodes
Triggering	Normally event triggered	Normally event triggered, time triggered possible but not reasonable	Time triggered
Medium access	A task sends a transmit command to the CAN controller which sends the message upon winning arbitration i.e. access to the bus	The slave sends a response to the request by master (time triggered thus possible with protocol master distributing its view of time)	Data is sent to a send buffer, the controller forwards it to the bus in the predefined slots

Table 2.1: Comparison of automotive communication systems

Table 2.1 does not include any information regarding unit prices and development cost of a system implementing the presented communication systems. However, the absence of open source network design software for FlexRay networks, without which the complexity of developing such a network would not be manageable, makes the development of such a FlexRay based system prohibitive. The added absence of development boards with built-in FlexRay transceivers and necessary circuitry would lead to further costs from the evaluation phase.

For the purposes of the elevator control system presented here, the following points are assumed:

- Initially, a total of six floors are to be handled by the elevator system as per the available elevator model,
- The elevator car travels at a maximum speed of 1 m/s,
- The system should be designed to allow for use in buildings up to 20 floors,
- One node is to handle each magnetic sensor,
- One node is to handle the incremental encoder sensor,
- One node is to handle the ultrasonic distance sensor,
- One node is to be used for the XINU interface and position evaluation,
- Expansion in terms of additional displays, floor and elevator car keyboards and similar units are to account for.

An approximate calculation of the number of nodes based on the above-mentioned assumptions leads to a total of 50 nodes connected to a network. The bus length in this case, based on an assumed floor height of 3 m and a total of 20 floors, amounts to approximately 60 m. For routing of the bus into the elevator car an additional 20 m would be required. The total bus length can be assumed to approximately 100 m.

The communication system of choice for this application would be CAN. At the mid-range bit rate of 125 kbp/s the bus length amounts to over 300 m, as shown by equation (2.1) and is equal to 500 m according to [CAN02]. This maximal bus length at the selected bit rate fulfills the required bus length and allows for significant additional expansions of the system.

$$buslength_{maximum} = \frac{1}{bitrate * totaldelay_{propagation}} \tag{2.1}$$

The propagation delay, assuming twisted-pair copper wires as the transmission medium, is about 5 ns/m. The total propagation delay is approximated as five times the propagation delay, in order to account for the protocol-specific requirement of arbitration. In order for the CAN concept of arbitration to function properly it is necessary that each node in the network recognizes the first arbitration bit and responds within one further arbitration bit. It has to be ensured that two nodes at the furthest ends of the bus have sufficient time to output their own and read the state of the other nodes, bringing another factor of two into the equation. An addition of 25% is provided as a safety margin giving the factor of 5 for the total propagation delay.

A CAN bit rate of 125 kbp/s and a maximum length of a CAN message of 120 bits (8 bytes of data payload and worst-case situation of 18 additional bits for bit stuffing) give a theoretical maximum of 1041 CAN messages per second of data throughput on the bus. Three CAN messages from sensors are required as inputs for the position calculation, giving the lowest period for position calculation of 2.88 ms. If the assumed maximum elevator car speed of 1 m/s is taken into account, the error of the calculated position due a communication bus amounts to 2.88 mm.

The calculation given above does not include delays caused by the arbitration process of CAN, delays caused by the CAN controllers and transceivers and delays caused by the software running on the sensors and on the controller that runs the position algorithm software, as well as the inertia of the electric motor and its coupling to the elevator car.

A comparable project based on an approach of using CAN as the communication bus for an elevator control system is seen as a departure from the classical concept of elevator control based on relays in favor of a system of interconnected programmable controllers [HKT09]. The change of focus and inclusion of software-based elevator control systems has enabled optimization schemes based on traffic prediction and traffic pattern recognition for elevator group control systems, as presented in [CLXC06] and [KCS00].

3 Methodology

A top-down approach illustrated in Figure 3.1 was followed as the development methodology during the requirements and design phases, starting with general system requirements and the recognized safety goal. Through an analysis of these requirements, a collection of safety and functional requirements is compiled. Further, a decomposition of the functional and safety-relevant requirements is carried out in order to reach the hardware and software requirements. The requirements serve as a basis for the system design stage, further leading to the implementation. In the first place they influence the selection of what functionality is assigned to hardware and what to software. In the second stage, the functionality to be implemented in hardware is assigned across the distributed system, as well as that which is to be implemented in software. For hardware this can mean that certain nodes in a distributed system need to feature specific interfaces and functionality. With software in mind, nodes with specific hardware features require specific corresponding software that makes use of these features. A further aspect of a split of software functionalities across a distributed system is the possibility of optimized use of resources. Certain resource-intensive modules recognized during the design stage can be mapped to an appropriate hardware node, taking into account any constraints with regard to timing, co-location with other modules or required hardware functionality.

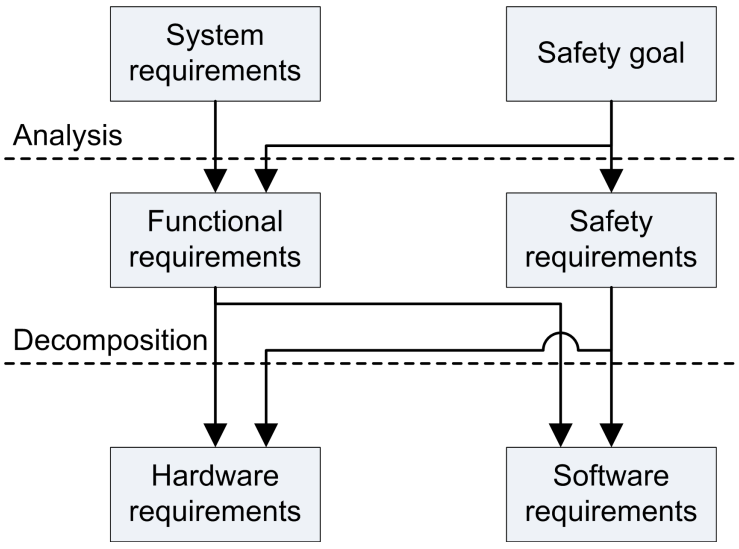

Figure 3.1: Top-down approach to system design

3.1 System requirements

Little has changed in terms of general functional requirements put on elevators since the first elevator featuring safety elements was introduced by Elisha Graves Otis in 1854. The additional requirements put on this particular elevator control system are influenced by the additional functionality that it needs to support.

Due to a large spatial distribution of the system brought about by requirements on optimal sensor placement with regard to measurement quality and additional logic needed at each sensor position, it can be freely assumed that a distributed system would be required.

A distributed system brings with it the need to bridge the gap between the discrete hardware parts, to solve which a communication channel is required. Additionally, the system needs to provide an interface to the control board that controls the movement of the elevator car. The XINU environment, as represented in Figure 1.3, serves as the management plane of the entire elevator control system. Therefore, an interface with the XINU environment needs to be provided as well.

The following general system requirements are to be taken into consideration during system design.

System Requirement 1 *REQ_SYS1_FUNCTION:* The system shall use the current position and movement information of the elevator car to instruct the motor control board how to reach the requested position.

System Requirement 2 *REQ_SYS2_HW_EVAL:* The system shall use available evaluation boards as the hardware platform.

System Requirement 3 *REQ_SYS3_COMMON_BUS:* The constituent hardware modules of the system shall be able to communicate using a common bus.

System Requirement 4 *REQ_SYS4_IF_MOTOR_CTRL:* The system shall provide an interface to the motor control board.

System Requirement 5 *REQ_SYS5_IF_XINU:* The system shall provide an interface to the XINU environment.

System Requirement 6 *REQ_SYS6_CMD_XINU:* The system shall accept call commands from the XINU environment.

3.2 Safety goal

One of the major goals that this elevator control system has to achieve is that of safety while in operation. It should not be possible for humans to come to harm through the functioning of the elevator, i.e. the movement of the elevator car. This further means that the elevator car should only then be allowed to move if its position can be accurately and unambiguously determined. In order to keep this goal in focus while developing the system it shall be defined, and is to be in focus of each design and development phase.

Safety Goal 1 *REQ_SAFETY_GOAL:* The system shall cause the elevator car to move only if its position can be unambiguously determined.

3.3 Requirements analysis and decomposition

Further design of the system is based upon the outlined system requirements and the safety goal. The safety requirements and the functional requirements are reached through an analysis of the system requirements and the safety goal and thus present a lower abstraction level of the system.

A decomposition of the obtained safety and functional requirements is necessary in order to reach the individual hardware and software requirements. The requirements obtained in this way are of sufficient detail to serve as a basis for implementation and shall be used directly in the implementation of the system hardware and software.

3.3.1 Safety requirements

By decomposition of the Safety Goal 1, the recognized safety goal, it follows that there are at least two states that an elevator car can be in. These two states are the safe state and the operational state, and are defined and described as follows.

The safe state is one in which the system does not pose a threat to persons or property, and one in which the system can stay in for indefinite periods of time. For the system presented here this state is static, i.e. an elevator car is then in its safe state when its movement stops and the car is not allowed to move further. Upon entering the safe state, the system has to cause the elevator car's movement to stop. The elevator car has to remain stationary as long as the system is in the safe state.

Definition 1 *Safe state:* The safe state of this system is the one in which the elevator car is and remains stationary.

The operational state is one in which the system can take commands from a management plane and initiate elevator car movement. While in the operational state, the system regulates the movement of the elevator car according to its current position and the received information on user wishes.

Definition 2 *Operational state:* The operational state of this system is one in which the movement of the elevator car can be initiated by the system according to user input.

Definition 3 *Init state:* The init state is the state which the system is in upon start-up and during which the components go through a process of initialization.

Figure 3.2 depicts the system states and the allowed transitions within the system. In normal operating conditions after a reset the system is in the init state where all the necessary initializations of hardware and software are to take place. As each node initializes its own internals it transits into the safe state. The master node, upon its initialization and transition to the safe

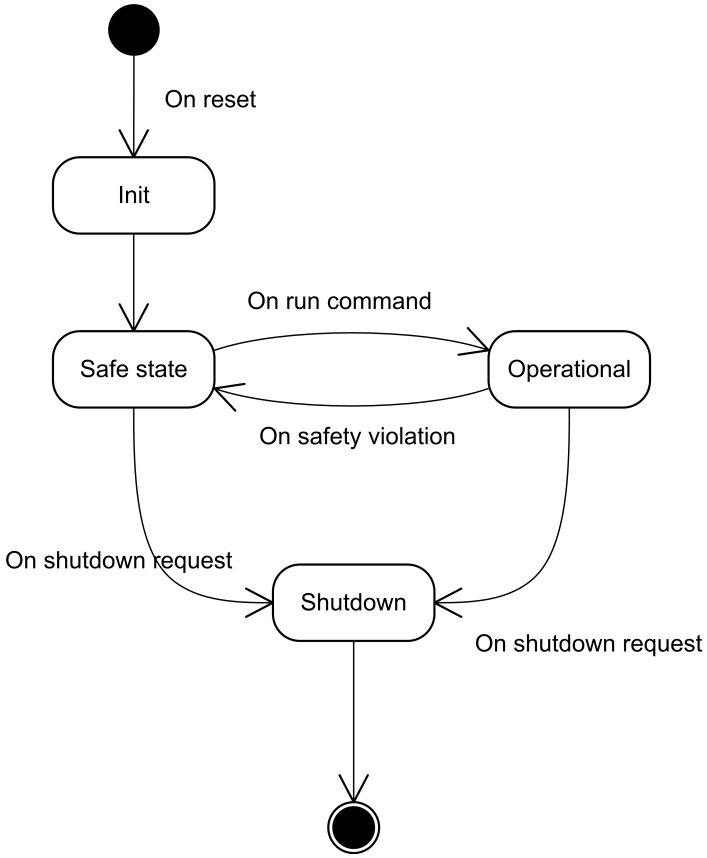

Figure 3.2: State diagram on the system level

state, evaluates the current position and input from the XINU environment. Several parameters are to be considered at this stage, such as whether all the nodes that provide the information necessary for a position calculation are present on the bus and responsive as well as whether a position value that is calculated is within the required limits of precision.

Based upon the outcome of the evaluation a decision is made whether a transition to the operational state can be allowed or the system should remain in the safe state. A run command from is issued triggering the transition of the system into the operational state.

Thereupon, the system remains in the operational state as long as neither a safety violation has occurred, necessitating a transition to the safe state, nor the master node has received a shutdown request from the XINU environment. Upon reception of a shutdown command the master node issues a system-wide shutdown command over the common bus, indicating that each node is to deinitialize itself and prepare for a shutdown.

As each node completes the deinitialization process it transits from the shutdown state to the off state automatically. The entire system is restarted by reconnecting the power supply, which causes an entry to the state diagram at the init state. This ensures that all the nodes go through the initialization process, are in the safe state and thus are ready to accept and act upon the run command of the master node.

A controlled way to shut down the elevator control system is given in the shutdown state. Once in this state, the system can be disconnected from the power supply to achieve a controlled shutdown with no risk of harm. It is the task of the master node to notify the other nodes on

the common bus that a deinitialization is to take place and no further communication is to be expected. Upon reception of this message each node is to yield release any resources reserved by it and cease communication. The execution of any communication protocol stack running up to this point is to be stopped and the peripherals turned off if possible.

Based on the Safety Goal 1 and the system states defined in this section, the following safety requirements can be identified.

Safety Requirement 1 *REQ_SAF_POS_PRECISION:* While the system is in the operational state, the position of the elevator car shall be calculated with a precision of ±4 mm.

For maximum passenger safety, the allowable precision of the calculated position value is defined in this application as twice the precision of the least precise sensor used in the system. In this system the least precise sensor, according to the information found in its datasheet is the ultrasonic sensor. Its overall precision as given by the manufacturer's datasheet is ± 2 mm, influencing the position precision required by the Safety Requirement 1 to be ± 2 mm as well.

Safety Requirement 2 *REQ_SAF_OP_TO_SAFE_TIME:* While the system is in the operational state, if the precision with which the position of the elevator car is calculated is not within ±4 mm the system shall transition into the safe state within 4 ms.

The maximum allowable reaction time in the case of a detected invalid position measurement is calculated using the allowable position precision and the expected maximum elevator car speed of 1 m/s.

3.3.2 Functional requirements

Several concepts and requirements can be derived from an analysis of the system and safety requirements.

The first concept that to be developed on the basis of the safety goal is that of the method to determine the elevator car position, or rather, to determine it unambiguously using the input from diversified sensors. Each of the three types of sensors chosen for the elevator control project in [Mai11], apart from requiring a different physical interface to the controller board, also features a different resolution and measurement dynamics.

One type of sensors already in use in the system is that of a magnetic field sensor. The sensor selected for this task offers impressive dynamic characteristics, such as very short turn-on and turn-off delays of under 0.05 ms. The overall resolution of this sensor depends on the floor height, but the resolution when the sensor is near the magnet is ±6 mm, as indicated by the calculation in [Mai11]. However, this sensor is intended to be used as a digital one, indicating that the elevator car arrived at a predefined point coinciding with a floor of the elevator shaft. Between two such positions there is a continuous range of elevator car positions that can not be detected using the magnetic field sensor. The discrete positions where the magnets are placed to mark a floor are to be points at which the position algorithm evaluates the data from all three sensor types.

Another type of sensors used in the system is an ultrasonic distance sensor. Such sensors typically have a blind area in which no measurement can be made. With the selected model this is the

area from 0 mm to 80 mm from the tip of the sensor housing. This unusable area does poses serious restrictions neither on the elevator model nor as any production system since the sensor can simply be located 80 mm further away from the zero-position. The ultrasonic distance sensor used features a resolution of ±1 mm, as given in the technical data of the sensor.

The third sensor type that finds use in the system is an incremental rotary encoder. In the elevator model it is mounted on the spindle that translates the rotary movement of the electric motor to the linear movement necessary to move the elevator car through the shaft. The encoder used provides 100 pulses of each quadrature signal per one spindle turn which, together with the linear travel of 4 mm per spindle revolution, calculates to a sensor resolution of 0.04 mm per valid pulse. A valid pulse consists of a recognized pulse on both the A and B tracks of the quadrature output of the rotary encoder, as illustrated in Figure 3.3. A third signal line, the Z track, is present as an indication of a zero-point of the rotary encoder. The rotary encoder is of a multi-turn relative position type, making the zero-point useful as a redundant way of sensing a full turn of the spindle and re-calibrate the pulse counters for tracks A and B at each turn of the spindle. The encoder itself does not provide an indication of an absolute position or a count of turns. Due to the additional imprecisions introduced by the mechanical parts, such as imperfect coupling of the electric motor and the spindle as well as slip of the elevator model car, the overall resolution of a measurement using the rotary encoder sensor amounts to ±2 mm.

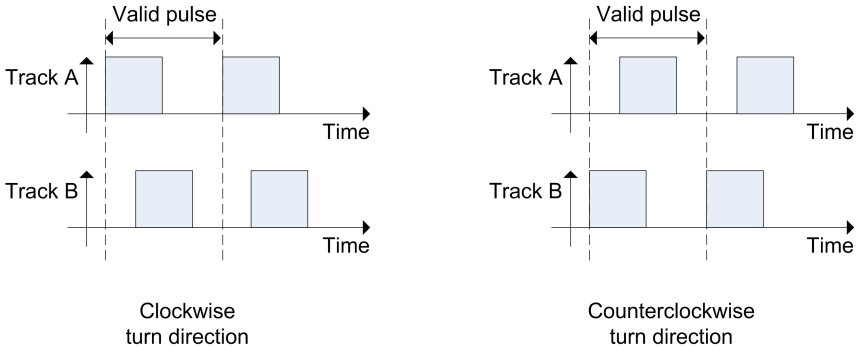

Figure 3.3: A valid quadrature pulse in the clockwise and counterclockwise turn direction

Two separate cases need to be taken into account when the position calculation is concerned. The distinction is necessitated by the significantly differing resolution values of the magnetic field sensor. Thus, two notions of position can be defined and are to be used in the further design and implementation stages.

Definition 4 *Discrete elevator car position:* The position of the elevator car expressed as the building floor number where one magnetic sensor is within range of the magnet mounted on the elevator car.

Definition 4 of the discrete elevator car position is important for the safety-related function of the system. It is in such positions of the elevator car when the safety-relevant algorithm is to be ran in order to verify the quality of the acquired position information from the three sensor types. The output of this algorithm is a confidence value which evaluates the deviations of the measured position values. This information serves as the basis upon which the master node is to decide whether the system is to be allowed to remain in the operational state or it a transition to the safe state is required.

Definition 5 *Continuous elevator car position:* The position of the elevator car expressed as the position in meters from the zero mark of the elevator shaft.

The definition 5 of the continuous elevator car position provides the position information in the range measurement range between two floors. This information can be obtained from only two of the three sensors and can thus not be used to directly influence the decision on transiting to the safe state. Nonetheless, it provides useful input for regular operation and additional error detection with information on the current speed and direction of movement of the elevator car.

An analysis of the requirements and definitions presented in this chapter yielded the following functional requirements.

Functional Requirement 1 *REQ_FUNC_CARPOS_VALID:* The system shall provide a mechanism to reach a binary decision on whether the current calculated value of elevator car position is valid.

In order to enable the transit into the safe state, the system needs to implement a mechanism that uses position information from the diverse sensors and evaluates whether an unambiguous position value can be calculated from them. The result has to be strictly a binary value, based upon which a state transition can clearly be made. This necessity is stated in the Functional Requirement 1, which was derived from the System Requirement 1 and the Safety Goal 1.

Functional Requirement 2 *REQ_FUNC_CARPOS:* The system shall use all three sensor types to calculate the current position of the elevator car.

Functional Requirement 3 *REQ_FUNC_CARSPEED:* The system shall use all three sensor types to calculate the current speed of the elevator car.

Functional Requirement 4 *REQ_FUNC_CARDIR:* The system shall use all three sensor types to calculate the current direction of movement of the elevator car.

The Functional Requirements 2, 3 and 4 upon which the system design is to be based were reached through an analysis of the System Requirement 1 using the defined elevator car positions defined in this section.

Functional Requirement 5 *REQ_FUNC_NET_CAN:* The system shall use a CAN network to connect its hardware modules.

As per System Requirement 3, a communication bus shall be implemented on CAN. Since the CAN specification covers aspects of the Physical and Data Link Layer described in the OSI Reference Model, as illustrated in Figure 2.2, a further analysis of software requirements is necessary.

Functional Requirement 6 *REQ_FUNC_IF_MOT_CTRL:* The system shall provide an interface to the motor control board with the physical layer based on the serial protocol RS-232 and a proprietary frame format.

The motor control board that directs the electric motor build into the elevator model uses the RS-232 standard for serial transmission of data as the physical layer. The data link layer is a thin proprietary protocol to exchange control data. Based on this data the motor control board causes the motor to spin in the requested direction or to be stationary. This functional requirement is derived from the System Requirement 4.

Functional Requirement 7 *REQ_FUNC_IF_XINU:* The system shall provide an interface to the XINU environment with the physical layer based on the serial protocol RS-232 and a proprietary frame format.

The XINU environment as well uses the RS-232 standard for serial transmission of data to the elevator control system. Another type of data link layer is in place to enable the exchange of data and recognition of commands. The basis for this functional requirement is the System Requirement 5.

Functional Requirement 8 *REQ_FUNC_CMD_XINU:* The system shall react on call commands from the XINU environment.

This functional requirement is directly caused by the System Requirement 6 and requires no further analysis at this point.

Functional Requirement 9 *REQ_FUNC_HW_EVAL:* The system shall use available evaluation boards with at least one CAN transceiver and RS-232 interface.

As per System Requirements 3 and 2, evaluation boards featuring bus access support as the communication bus shall be used. Functional Requirement 5 states that the bus to be used is CAN, and it follows from Functional Requirements 6 and 7 that RS-232 as a physical layer needs to be supported as well.

3.3.3 Hardware requirements

The previously defined safety and functional requirements are analyzed, along with any constraints on the system, hardware and software. From this analysis a number of requirements on hardware and software follow, which serve as the basis for the design of the presented elevator control system.

Most of the identified requirements up to this point have been of a general nature, not attempting to influence the breakdown of functionality and mapping to specific hardware and/or software. From this point in the analysis onwards, decisions are made that influence the system layout and the mapping of the required functionality to specific parts of the system. The final mapping of functionality to specific nodes in the distributed system, used as the basic design of this system, is described in Chapter 4 on System design.

Hardware Requirement 1 *REQ_HW_DIG_IN:* The system shall provide an interface for reading the state (digital level) of an input line.

The Hardware Requirement 1 follows from Functional Requirements 2, 3 and 4, which require the support of reading the output state of a magnetic field sensor. The magnetic field sensor used in the system provides an essentially digital output which needs to be interfaced to the hardware and its status read.

Hardware Requirement 2 *REQ_HW_PWM_IN:* The system shall provide an interface for measuring the frequency and duty cycle of a PWM signal applied to an input line.

The Hardware Requirement 2 is also derived from Functional Requirements 2, 3 and 4, which require the support of measuring the characteristics of the PWM signal provided by an ultrasonic distance sensor. The length of the positive signal pulse is related to signal frequency and the nominal sensing range of the sensor in use to arrive at the value representing the distance of the object to the ultrasonic sensor.

Hardware Requirement 3 *REQ_HW_QUAD_IN:* The system shall provide an interface for measuring the amount of pulses of a quadrature-encoded signal applied to two input lines.

The Hardware Requirement 3 is the third hardware requirement derived from Functional Requirements 2, 3 and 4. Support of measuring the three-line quadrature output signal, providing relative rotational information of a rotary encoder, is required. Signals output on lines A and B carry the information that the encoder has just passed on of the defined position points along a circle, with the direction of movement given by the phase shift of the two pulses. Only movement of the encoder shaft produces output pulses; otherwise the lines retain the last stable value. Signal output on the Z line indicates passing of the zero-mark, indicating a full revolution of the shaft. Thus, a minimum of three input pins are required at the target hardware.

Hardware Requirement 4 *REQ_HW_IF_CAN:* The system shall use hardware elements, each of which provides an interface compatible to the CAN specification and a compatible line driver.

In order to cover the distances within a building, parts of an elevator system need a robust communication network. The Controller Area Network was chosen for this purpose, as described in Chapter 2.5. Each hardware node of a distributed system thus needs to have support for communication over CAN. Current state of the art sees a departure of discrete CAN controllers connected through serial interfaces, in the form of CAN controllers integrated in an MCU. The CAN specification [Bos95] does not provide fixed specifications for CAN line drivers, although it does note that all nodes in a CAN network inherently need to be using compatible medium access drivers in order for communication to be possible. For practical purposes medium access has been standardized and published in the ISO standard [ISO03] and compatible CAN transceivers are readily available from numerous manufacturers.

Hardware Requirement 5 *REQ_HW_IF_MOT:* The system shall provide an interface compatible to the RS-232 physical layer for communication with the motor control board.

A control board, addressable by RS-232, is provided for control of the DC electric motor movement. Thus, the system needs to provide an RS-232 interface and support the thin protocol to be used for communication with the motor control board.

Hardware Requirement 6 *REQ_HW_IF_XINU:* The system shall provide an interface compatible to the RS-232 physical layer for communication with the XINU environment.

A computer running the XINU environment, addressable by RS-232, is provided for higher-level functions such as elevator car call control. Thus, the system needs to provide an RS-232 interface and support the protocol for communication with the XINU environment.

Hardware Requirement 7 *REQ_HW_EVAL:* The system shall use available evaluation boards, each containing a CAN transceiver compatible to ISO 11898 and an RS-232 interface.

In order to avoid high costs of producing custom hardware for this elevator control system and to be able to use as many standard off-the-shelf and open source components an evaluation board is to be selected as basic hardware for system nodes. An important constraint for the selection process would be the possibility to fulfill and compatibility with the other identified hardware requirements.

3.3.4 Software requirements

Apart from being the basis from which hardware requirements are derived, the safety and functional requirements defined in this chapter are the foundation for requirements on software. The previously derived requirements on hardware influence the software as well; however this relation is described in Chapter 4 on System design as the derived hardware requirements are of a level with sufficient detail to be used in the design.

Software Requirement 1 *REQ_SW_POS_PRECISION:* The software shall implement an algorithm that reaches a binary decision on whether the elevator car position is valid. Validity is in this application defined as sensor values that are within the precision of ±4 mm of each other.

Through combination of the Safety Requirement 1 and the Functional Requirement 1, Safety Requirement 1 is derived. An algorithm is to be developed that uses position values obtained from the sensors to calculate a validity of the measured position. A binary output that can be used as input for other software modules is expected.

Software Requirement 2 *REQ_SW_POS_REACTION_TIME:* The software in the operational state shall, upon detecting an invalid position, transit into the safe state within 4 ms.

Software Requirement 2 follows directly from the Safety Requirement 2 and describes the maximum amount of time that can be allowed to pass if the system is in the operational state and an invalid elevator car position is measured. In a distributed system such as the one being developed this requirement implies constraints on system layout. For example, it would be practical to avoid delays induced by using a communication medium to transfer information between a node hosting the algorithm that evaluates the position precision and a node controlling the motor control board. This design decision would also help reduce the jitter cause by the communication network and make verification of this software requirement more practical.

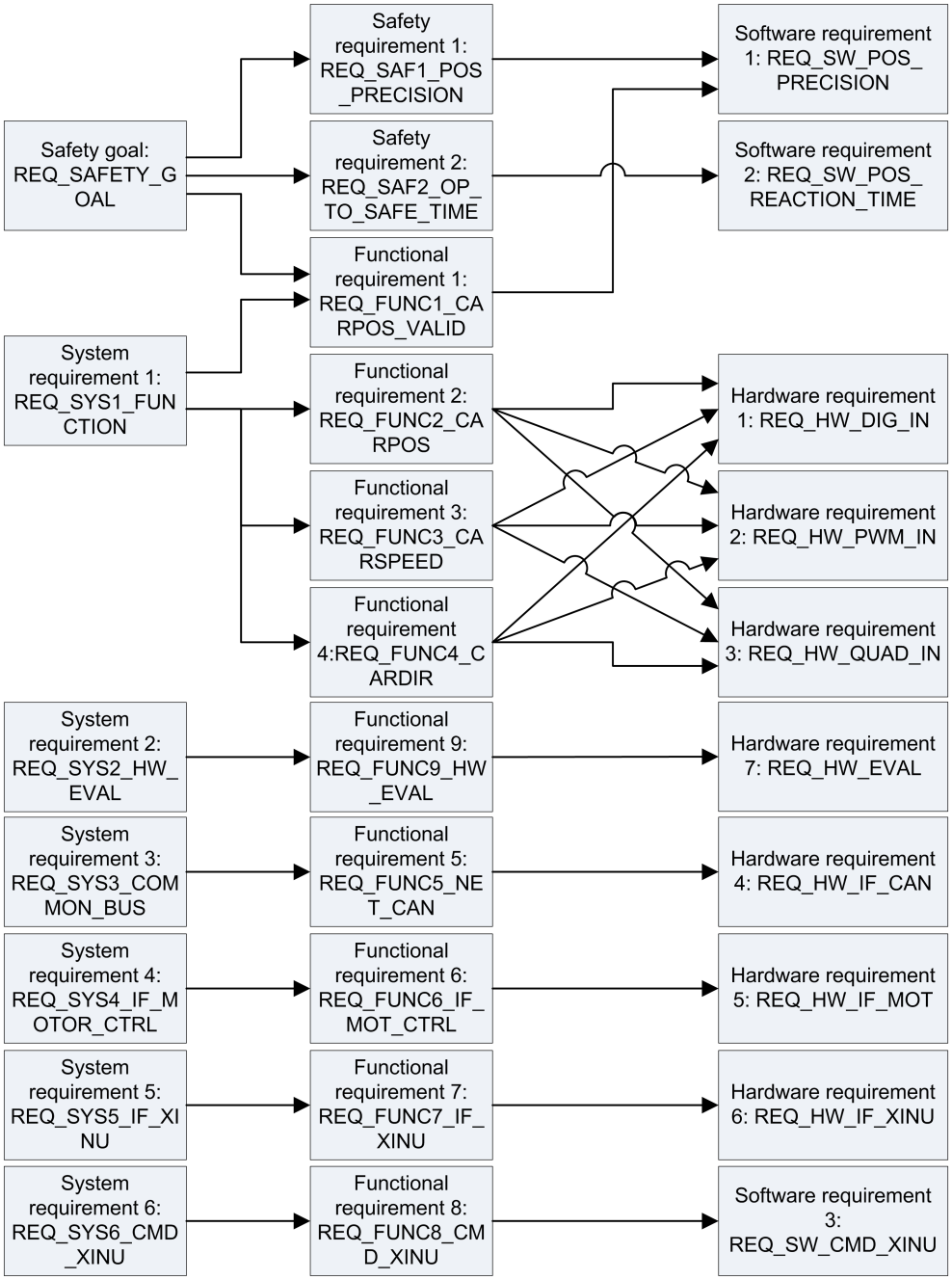

Figure 3.4: Tracing of system requirements and the safety goal onto functional anf safety requirements

Software Requirement 3 *REQ_SW_CMD_XINU:* The software shall accept instructions from the XINU environment to instruct the movement of the elevator car.

In order to enable communication with the XINU environment, apart from hardware support for RS-232, support on the software for the packet framing used by XINU is also required. Since communication with XINU has not been identified as safety-relevant, decisions on mapping this communication functionality to even a general node in the system are not made at this point to enable flexibility in the design stage.

3.3.5 Traceability

In order to aid traceability, Figure 3.4 illustrates the relationships between the System Requirements and the Safety Goal as the most general ones, the Safety and Functional Requirements derived from them, and the Hardware and Software Requirements representing the input for the design stage of development.

This traceability matrix is meant to help ensure that all the input requirements, as stated in the System Requirements and the Safety Goal, have been mapped further onto the Safety and Functional Requirements, and onward to Hardware and Software Requirements. This relation of forward traceability is important in order to ascertain that the input requirements are properly analyzed and taken into account in the design stage. The relation in the opposite direction, backward traceability, helps avoid features that would appear at the design stage and influence the design, but could not be traced to an original higher-level requirement. Such additions can be viewed as ghost features that are not to be part of the design.

4 System design

The goal of this work is to specify and design a part of an elevator control system that uses diversity to raise fault detection. Diversity on the system level is to be achieved through use of two independent communication systems with different physical and protocol properties. Diversity on the board and application level is to be achieved through use of different processors, peripherals as well as the independently developed software and hardware. Diversity on the sensor level is given by the choice of sensors as described in [Mai11]. The selected sensors utilize diverse physical properties to obtain the measurement values, and have disparate resolution and accuracy values. Further diversity is influenced by the choice of microcontroller units for processing, which pose different requirements on interfacing with the sensors as well as influencing the way functionality is mapped onto hardware and software. Diversity on the software level is influenced by the choice of fundamentally different processor cores running on the boards of each system. The software is to be designed and implemented by independent persons that should communicate strictly to exchange interface data, e.g. data on the access of commonly used sensors and common software interfaces, and not on design decisions.

This chapter presents the design phase of the development and the reasoning behind the choices made for both hardware and software. Decisions regarding hardware are made concerning how to interface sensors of various specifications and a number of RS-232 compatible devices and systems to the development boards. With regard to software, decisions concerning the software architecture, communication stack and development methods and tools used to design the system are made and described.

For this purpose, hardware and software requirements are used as input documents and serve as a basis for further system design. In the first place they influence the way that the required functionality is assigned to hardware and software, and further the way the functionality is distributed between the constituent components of hardware and those of software.

4.1 Hardware design

As with most applications in building automation, elevator control systems have restrictive spatial constraints as well. In order to use sensors in an optimal way they might need to be placed far away from other sensors or a centralized controller. Layout of such a system is influenced to a large degree by considerations of retaining integrity of signals coming from sensors and bearing important information, such as in this case the position of an elevator car.

In order to retain signal integrity the controllers that process raw sensor data should be placed as close to the sensors themselves as possible. In a system with elements spread far apart, a centralized solution with one controller comprised of hardware to interface with each of the sensors, a CPU and device driver, communications and applicative software running on this CPU presents significant challenges.

In designing this elevator control system, the approach of designing a distributed system was chosen. Conforming to the hardware requirements recognized presented in subsection 3.3.3, a solution using an already available basic hardware platform in the form of an evaluation board is to be chosen. In line with the other hardware requirements, the evaluation board has to provide the following:

- Interface for reading a digital input,
- Interface for measuring of a PWM input signal,
- Interface for measuring a quadrature-encoded input signal,
- Interface for communication over CAN,
- Two interfaces for communication over RS-232.

The evaluation board chosen as the platform board of this distributed system is the board produced by OLIMEX Ltd with the designation "STM32-P103". The CPU integrated onto the board is an STM32F103 microcontroller by STMicroelectronics, build around an ARM CORTEX-M3 core, capable of execution at up to 72 MHz, and supplemented by various peripherals such as ADC and DAC units, timers, CAN, USB and serial communication controllers.

The evaluation board chosen as the hardware platform in the system, and the integrated microcontroller, apart from other features and peripherals, provides support for all of the functionality required by the hardware requirements. The following Hardware Design Artifacts have been derived from the hardware requirements:

Hardware Design Artifact 1 *DES_AIM_HW_EVAL:* Use of available "STM32-P103" evaluation boards,

Hardware Design Artifact 2 *DES_AIM_HW_DIG_IN:* One "STM32-P103" evaluation board with six header pins that are connected to the microcontroller pins with digital input functionality shall be used to interface with six magnetic field sensors,

Hardware Design Artifact 3 *DES_AIM_HW_PWM_IN:* One "STM32-P103" evaluation board with two header pins connected to microcontroller pins with PWM input functionality and one header pin connected to a digital input pin of the microcontroller shall be used to interface with the ultrasonic distance sensor,

Hardware Design Artifact 4 *DES_AIM_HW_QUAD_IN:* One "STM32-P103" evaluation board with one header pin that is connected to microcontroller pins with functionality to interface with a quadrature encoder shall be used to interface with the rotary encoder,

Hardware Design Artifact 5 *DES_AIM_HW_IF_CAN:* Each of the "STM32-P103" evaluation boards includes a connector providing access to the ground, CAN-high and CAN-low lines of the CAN driver, further connected to the CAN controller lines of the microcontroller,

Hardware Design Artifact 6 *DES_AIM_HW_IF_RS232:* One "STM32-P103" evaluation board that includes a connector providing access to one UART channel over a RS-232 driver and access to the pins of a separate UART controller of the microcontroller shall be used to interface with the XINU environment and the motor control board.

This breakdown of hardware design artifacts, apart from presenting a way of further tracing the hardware requirements of the previous chapter, represents a foundation for low-level design decisions such as mapping of certain required functionality onto hardware modules supported by the "STM32-P103" evaluation board and its microcontroller.

For the purposes of this project, where an elevator model with a total height of less than 2 meters is used to represent a 6-floor building, one evaluation board is to be used to read the current state of all 6 magnetic field sensors. The decision was influenced by the amount of available evaluation boards. However, this design choice does not hinder system performance in any way since the use of a single evaluation board for all 6 digital inputs represents a valid model of the CAN bus load as well as presenting no penalties on the execution time of the CPU handling the reading. Load of the CAN bus is not influenced since the events on which messages relating to magnetic field sensor data are triggered are very sparse and far apart, both spatially as well as temporally. Furthermore, the bit-rate of the CAN bus was selected in such a way to accommodate an even larger amount of messages transmitted over it. The penalty on CPU load brought about by sampling 6 digital input registers by a single microcontroller is negligible, especially taking into consideration the absence of further processing required of this node, and is thus acceptable. As many of its contemporaries, the microcontroller used supports read operations on complete ports that consist of up to 16 pins each. The pins belonging to each port are mapped into adjacent bits in memory, thus forming memory locations addressable as 16-bit unsigned integer variables.

Among other functionality, the STM32P103 provides hardware support for interfacing with quadrature encoder signal pairs through six of its timer units. Only predefined pairs of timer unit input lines can be used for this purpose, with each timer unit provided with one input line pair. The encoder is in principle a clock with selectable tick direction. Several properties of the encoder interface mode can be configured, such as the active edge and whether the functionality is triggered of one or both input signals. Input of the zero-track signal is not supported in the timer hardware itself. To this end the zero-track output of the rotary encoder is to be connected to a general-purpose digital input pin. All general-purpose digital pins of the STM32P103 can be used to generate an interrupt on an external event. Use of an external interrupt source is acceptable in this application since the interrupt frequency in normal operation is inherently bound by the rotational speed of the spindle. A boundary case can be identified in the form of a quasi-stationary position of the elevator car and the internal zero-mark indicator coming to rest exactly adjacent to its detector. In such circumstances it is conceivable that the internal zero-mark indicator might be caused to toggle due to vibrations in the system. This would would further influence the zero-track output and cause it to take the form a pulse train. The described behavior has a negligible impact on the precision of the position measurement algorithm as the imprecision caused by this jitter was already taken into consideration when estimating the overall resolution of the rotary encoder and its corresponding mechanical parts and linkages. In order

to counteract any negative impact that the aforementioned behavior could on the load of the CAN bus, measurements in the software executed on the rotary encoder sensor node are needed. Appropriate measures would be a software-based debouncing function to handle a rapidly toggling zero-mark, time-triggered transmission of the position value from the rotary encoder or a debouncing function built into the communication software that would implement a minimum timeout between two transmissions.

Several timer units of the STM32F103 microcontroller support PWM input functionality in hardware. The PWM input mode is a specific case of the input capture mode one input, carrying the PWM input signal, is mapped to two internal input capture channels reacting on opposite signal edges. One of the input capture channels, channel IC2, is then used to capture the length of a signal pulse, while the other capture channel, channel IC1, captures the period of the signal and resets the counter of the first capture channel. In this way the counter value of channel IC1 represents the signal period in timer clock ticks, while the counter value of channel IC2 represents the duty cycle of the same signal. The counter registers are shadowed and provide the period and duty cycle measured in the last complete signal period.

An ISO-11898 [ISO03] compatible CAN transceiver and all of the circuitry required for its operation are already included on each STM32-P103 evaluation board. The transceiver is connected to the corresponding pins of the CAN transceiver integrated into the STM32F103 microcontroller. The proper configuration of CAN controller and pin settings is done through software.

Each STM32-P103 evaluation board provides includes one RS-232 driver interface that is further connected to the UART2 controller unit integrated into the microcontroller. Transmission and reception lines of a further controller, UART1, along with a signal ground line are lead through the extension header of the evaluation board. In order for the same evaluation to be used for communication over both of the required RS-232 interfaces, an adapter with a driver compatible with RS-232 would need to be used. This is the preferred solution that would help minimize the latency caused by use of two separate evaluation boards. In a two-board solution, the first evaluation board would be used to execute the elevator control software and to communicate with one RS-232 device, either the XINU environment or the motor control board, and the evaluation board communicating with the other RS-232 device. The other evaluation board would simply present a bridge between the other RS-232 device and the main evaluation board. Such a solution would cause delays caused by the added bridging of data from RS-232 to CAN, as well as jitter mostly caused by the arbitration mechanism of CAN used for medium access. Since the performance of the elevator control algorithm would significantly compromise the ability of the control algorithm to fulfill the Safety Requirement 2. Thus, the preferred solution is to use one evaluation board with its RS-232 interface to connect to the motor control board, and use an UART-to-RS-232 adapter to connect the same board to the XINU environment.

The STM32 reference manual [STM10] includes a description of the provided functionality and the settings necessary to configure the required hardware peripherals, as well as some of their common use cases.

The Hardware Design Artifacts presented in this section and the hardware peripherals they are mapped to do not completely fulfill the Hardware Requirements presented in subsection 3.3.3. In order to fulfill the Hardware Requirements, following changes to the hardware and additions in terms of adapter interfaces are necessary:

- Adapter to convert the magnetic sensor output signal voltage range to a range compatible with the general purpose input pins of the microcontroller,

- Adapter to convert the ultrasonic sensor output signal voltage range to a level compatible with the timer unit input pins of the microcontroller,
- Evaluation board used to measure rotary encoder sensor data is to be adapted so both input lines connected to the A and B tracks of the rotary encoder are connected to header pins,
- Adapter to convert the rotary encoder sensor output signal voltage range to a level compatible with the timer unit input pins of the microcontroller,
- Adapter to convert between the TTL voltage levels of the UART controller and the voltage levels of the RS-232 interface.

The identified changes to the hardware are of a general nature; an analysis of the particular changes to the evaluation board and layouts of interface adapter boards are presented in Chapter 5 on System implementation.

In order to properly interface with magnetic sensors, a voltage shifter would be required. The magnetic sensor requires a supply voltage of between 10 V and 30 V and is supplied from a 12 V power supply common to other sensors, with 12 V being the highest level of the output voltage. The STM32F103 includes numerous general-purpose input/output pins that are tolerant to input voltages as high as 9 V and such pins are used to interface to the magnetic sensors. To this end an interface board for each of the six magnetic sensors in the elevator model is to be made, containing a circuit that shifts the maximum voltage level of the sensor output from 12 V down to the nominal voltage of 5 V as pictured in Figure 4.1.

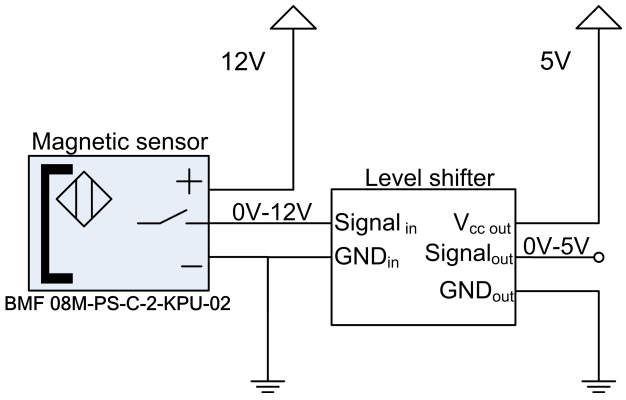

Figure 4.1: Layout of the magnetic sensor interface board

Figure 4.2 illustrates in a black-box fashion a circuit necessary to interface with the rotary encoder sensor. For optimal measurement performance, the two outputs A and B of the rotary encoder are to be connected to two specific timer pins of the STM32F103 that can be used in encoder interface mode, while the output Z is to be connected to a general-purpose pin and used to generate interrupts. The timer pins are to input voltages of up to 4 V and therefore need an interface circuit that would bring the voltage levels of the connected sensor to acceptable levels. The general-purpose input used to connect the Z track would not actually require such a circuit. However, the Z track will also be taken into account for the sake of uniformity and design clarity and its input will also be brought to the maximum level of 3.3 V, as well as the inputs that react to pulses on tracks A and B.

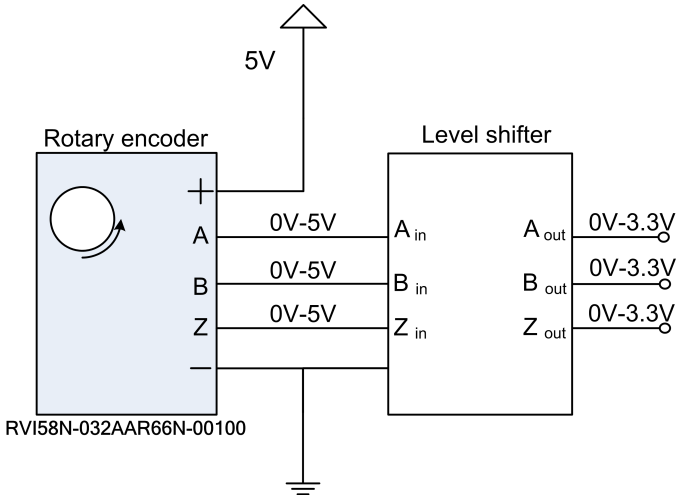

Figure 4.2: Layout of the rotary encoder interface board

A black-box view of an interface needed to connect the ultrasonic sensor is presented in Figure 4.3. The meaning and proper interfacing to the parametrization and synchronization inputs is discussed in section 5.1.1 on hardware interface adapter boards.

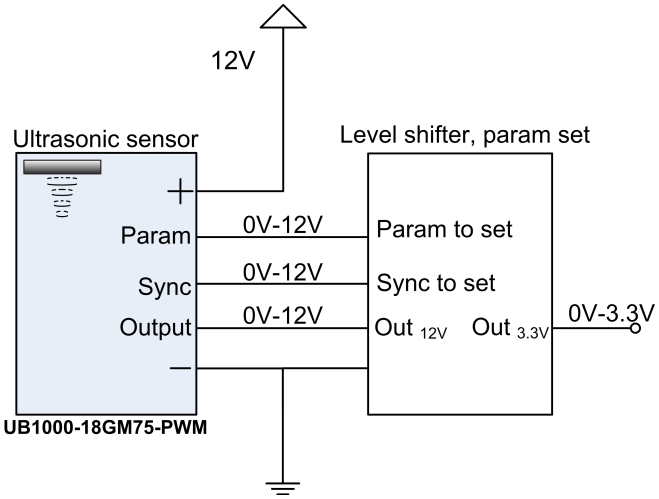

Figure 4.3: Layout of the ultrasonic sensor interface board

4.2 Software architecture

In line with the distributed approach chosen at the system and hardware levels, the accompanying software needs to be designed according to such an approach as well. According to the software requirements described in subsection 3.3.4, the following functionality needs to be provided by the software:

- Algorithm to reach a binary decision on the validity of the elevator car position,
- Transition from the operating state into the safe state is to happen within 4ms,

- Instructions from the XINU environment are to be used to influence the movement of the elevator car.

The following functionality is required to enable use of the hardware specified in the Hardware Design Artifacts of section 4.1:

- Software interface that provides the state of a digital input signal,
- Software interface that provides the measured properties of a PWM input signal,
- Software interface that provides the measured property of a quadrature-encoded input signal,
- Software interface that provides access to the CAN bus for transmission and reception of data,
- Software interface that provides access to the RS-232 interface for transmission and reception of data.

Further, a communication stack that would operate using the CAN controller is to be selected and integrated into the software of each evaluation board.

The software architecture used to design the master node software is shown in Figure 4.4. It is represented as a layered architecture that has its foundations in the hardware abstraction layer, which is in the case of the master node provided by ChibiOS/RT. ChibiOS/RT further provides the underlying operating system that is implemented as a preemptive kernel with a static architecture and resources allocated at compile-time [3]. Apart from the parts of the hardware abstraction layer needed by the operating system itself, the module CAN, GPT, UART and PORT are configured as they are required by the modules of the services layer, and ultimately by the application modules themselves.

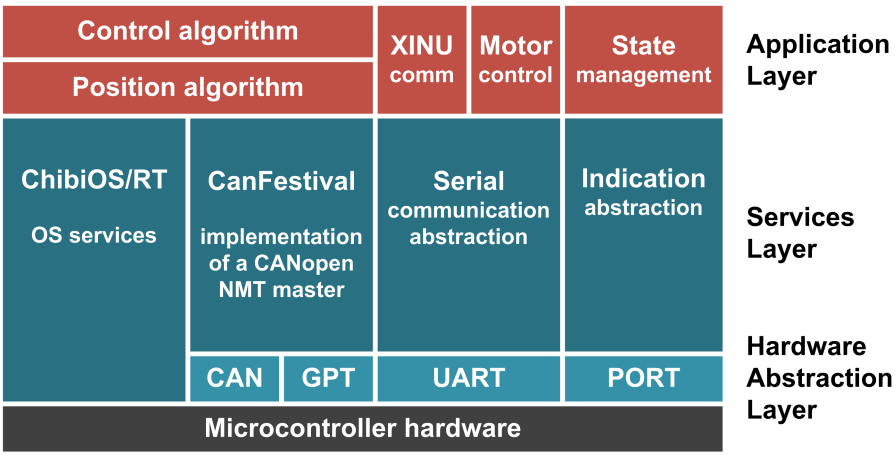

Figure 4.4: Software architecture - master node

The software architecture used to design the software of slave nodes is shown in Figure 4.5. Since the complexity of the software that is to be run on slave nodes is much lower compared to the master node, a real-time operating system is not required. Instead of a task scheduling module a run to completion scheme is used to implement the required functionality. The STM32F10x

Standard Peripherals Library, available as-is from ST Microelectronics [5], is used as the hardware abstraction layer. Any interface adaptations between the components, e.g. the CANopen communication stack, and the hardware abstraction are to be implemented as interface wrappers. Such wrappers can include handling of CAN transmission and reception, access to digital I/Os and timer units of the MCU.

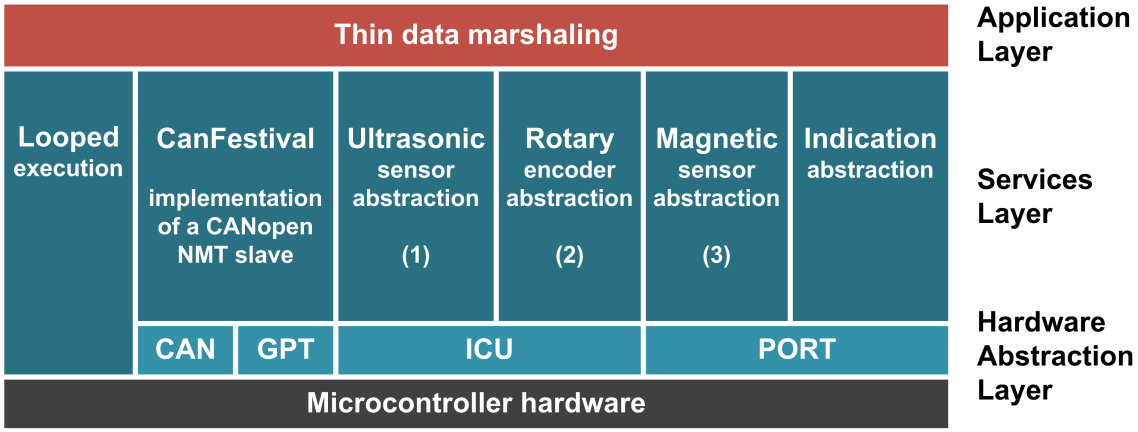

Figure 4.5: Software architecture - slave nodes

4.2.1 CANopen as the communication stack

The first communication system used in the presented redundant elevator control system, described in [Mai11], uses LonWorks as the underlying communication system. The second communication system, described in this thesis, uses CAN as the communication system. In difference to LonWorks, CAN specifies only the two bottom-most layers of an ISO-OSI communication stack, while the layers above and including the Network Layer are application specific, as shown in Figure 2.2. While it would be possible to use the eight byte payload field of a CAN message to directly transfer data, it would not be practical in this case. Such an approach is sufficient and efficient when used in static systems that do not need to be expandable. However, an elevator control system would benefit from a solution that would allow it to expand with the least amount of additional effort, for example by a reconfiguration of the system instead of a new development. Additional benefits to be sought from a communication stack used in a building automation system, or indeed any control system, are a heartbeat functionality that would help system integrity, as well as an implementation of controlled network startup and shutdown.

Two communication stacks that are designed to function using CAN as the bottom two ISO/OSI layers and offer similar services and functionality were investigated for this purpose, namely CANopen and DeviceNet. While both communication stacks are designated as 'Open', information on DeviceNet is significantly harder to obtain. Most information on CANopen, on the other hand, is openly and freely available, although the information on e.g. recently specified device and application profiles is typically not given open. There are several open source implementations of CANopen that are developed and supported by Internet communities and one of these was selected as the communication stack for the CAN-part of the elevator control system.

CANopen is a field-level communication protocol for distributed applications in industrial automation specifying services that implement an object-oriented distributed environment for system integration [BSCF98]. CANopen adds the notion of a node address or ID by reserving 7 bits

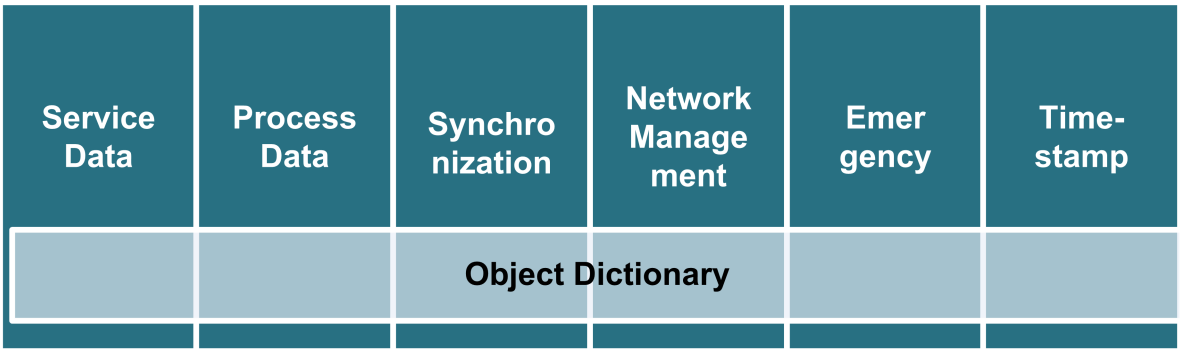

Figure 4.6: CANopen architecture

of the CAN message identifier for this purpose, while the other four bits represent the function of the message [PNH04]. The protocol is flexible enough to support use in diverse industrial applications, from multi-tap auto-transformers [PNH04] to automatic pipeline welding systems [WXC+08], with investigations into implementing CANopen for control of a hybrid electric vehicle [LHR+09]. The examples cited here do not limit the applicability of CANopen to other fields of use as many general and flexible device profiles exist that can, through combinations and adaptations, cover a wide field of possible applications.

Protocols of the CANopen stack, as specified in [CAN02], are displayed in Figure 4.6. The main concept of CANopen is the Object Dictionary, which is a data structure that serves as the foundation for the behavior of all communication objects, the state machine and process activities of a CANopen stack. Table 4.1 gives an overview of the areas that an object dictionary consists of. In the first place, an object dictionary specifies the data types used, most of which can be mapped into a PDO with the exception being data types describing strings. Scalar data types are specified as present in many programming languages, as well as ones describing time, communication parameters and manufacturer and profile specific values. An object dictionary further contains entries related to communication, such as the communication cycle period, hardware and software versions, period of the produced heartbeat and the expected periods of heartbeats to receive, as well as information on how transmission (TDO) and reception (RDO) data objects are to be mapped. The entry range 0x2000-0xBFFF enables support for manufacturer specific entries, as well as entries relating to specified device profiles and interface profiles.

For example, the device profile "CiA 401", entitled "Device profile for generic I/O modules", can be used in magnetic sensor nodes to propagate the current state of a magnetic sensor over CAN. The object dictionary of the magnetic sensor node would thus include entries specific to "CiA 401" devices; in this example it would present facilities of a digital input module that are to be entered into the entry at index 0x6000. Furthermore, entry entitled "TPDO 1", i.e. the primary process data object the node transmits, is to be configured in entry with the index 0x1800. This entry contains information on the CAN ID to be used for this message, the transmission type (cyclic, acyclic, synchronous, asynchronous or on-request), the inhibit time which is a higher-level implementation of de-bouncing functionality, as well as the event timer which is used for time-triggered transmission of PDOs. Specifics on the configuration of the object dictionary are described in detail in [CAN08] and [CAN02].

Furthermore, an object dictionary contains declarations of standard and device or manufacturer specific simple and complex data types, communication parameters of the CAN network, information related to the class the device belongs to as well as information that describes the device

Index	Entry description
0x0000	Not used
0x0001-0x025F	Data types
0x0260-0x0FFF	Reserved
0x2000-0x5FFF	Manufacturer specific
0x6000-0x9FFF	Device profile specific
0xA000-0xBFFF	Interface profile specific
0xC000-0xFFFF	Reserved

Table 4.1: Structure of the Object Dictionary

and its manufacturer.

With regard to communication, CANopen recognizes these three types: master-slave, client-server and communication based on the producer/consumer model. Each protocol uses one of these principles, according to applicability.

Service Data Objects (SDO) provide services which are mapped to appropriate areas of the object dictionary of the device in question. Service data objects are typically used to set device parameters and are as such not used for transport of process-related data. This property makes them a good example of client-server communication within CANopen. Service data objects are typically used in combination with highly configurable nodes. An SDO client that has access to the required parameters would connect to the appropriate SDO server and transfer the configuration data in a connection-oriented way. Such a configuration would include setting the parameters of the other protocols, except for network management, and it is thus more suited for use in the network management state STOPPED or PRE-INITIALIZATION.

Process Data Objects (PDO) are an example of the producer-consumer model. The configuration of sampled data to process data and further to CAN messages is defined either at compile-time or is distributed at network start-up by SDO. The actual distribution of process data is based on this configuration and follows the producer-consumer model. The producer packs the sampled data and sends it in the form of a PDO without expecting an acknowledgment from a receiver. A consumer, i.e. a node that includes the PDO in question in its object dictionary as a PDO to be received, receives the CAN message and forwards it to the CANopen stack running on its CPU. The CANopen stack handles the interpretation of data and further sets appropriate entries of the object dictionary to corresponding values that were received. An application that uses the data would then read the new values from the object dictionary.

The Network Management protocol (NMT) is a master-slave based service which allows the network master node to influence the network management state of the slaves nodes in its network. One device in the network is required to implement the NMT master functionality, and the master changes its own state using local services as implemented in the specific stack. The NMT specified four possible states that an NMT slave node can be in: STOPPED, PRE-OPERATIONAL, INITIALIZING and OPERATIONAL, as visible in the state diagram 4.7. Through the definition of NMT states, network management implements a way for network-wide synchronized start-up of communication, as well as application functionalities. All slave nodes are required to stay in the state STOPPED after start-up. The NMT master node would, after its own successful start-up and initialization, broadcast a command for the nodes to enter the PRE-OPERATIONAL state

i.e. to initialize their communication parameters and prepare for normal operation. A request send by the NMT master to enter the OPERATIONAL state indicates to the slave nodes that their application functionalities can be started, thus starting transmissions of Process Data Objects and normal system operation. Further functionality that is a part of Network Management are the two error control services, Life Guarding and Heartbeat.

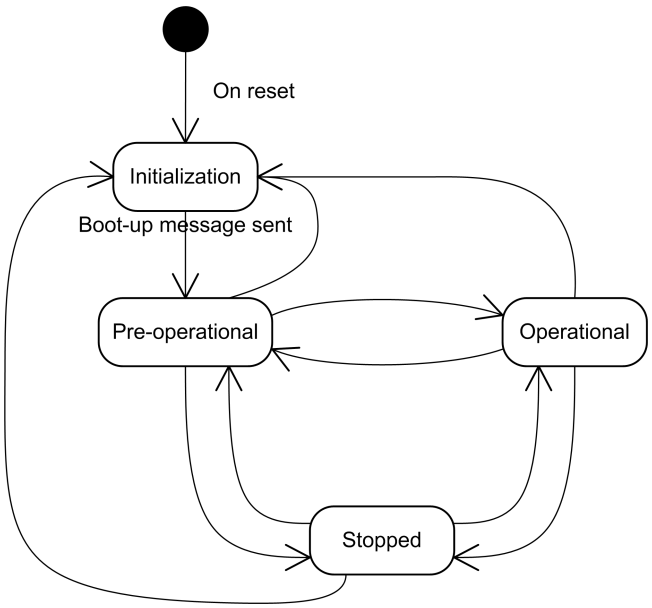

Figure 4.7: Network management state diagram

The Synchronization protocol is a non-mandatory producer-consumer based confirmation-less service which can be used to provide the notion of a common network clock. Synchronization messages are given a very low fixed CAN identifier of 0x80 in order to reduce the latency of distribution of common time caused by bus access. The messages themselves carry no data and can be interpreted as ticks of a common clock that are distributed over CAN. This service is used to coordinate synchronous behavior of devices, e.g. process control loops, with the process period being the period between synchronization messages that needs to be configured in the object dictionary of each affected node.

The Time-stamp protocol is another non-mandatory producer-consumer service which also does not need a confirmation from the consumers. In contrast to the synchronization service, Time-stamp messages using the CAN identifier of 0x100 carry the current time as a 6 Byte object and provide the notion of a common system time. The time distributed using the time-stamp protocol has the resolution of 1 ms and can have latencies higher than those of synchronization messages due to the CAN identifier of a comparably lower priority.

The Emergency protocol is a producer-consumer based optional service that provide the possibility to communicate an error by the producer in order to trigger a reaction by the consumers. The errors are device specific, although the CANopen specification [CAN02] provides a number of commonly occurring error classes.

Three of the six protocols specified by CANopen are relevant to this system and will be used; the remaining three, namely the Synchronization, Emergency, and the Time-stamp protocols are out of scope. However, these protocols specify functionality possibly important to other systems,

especially the Synchronization and Time-stamp protocols with regard to more complex distributed business logic that would require explicit time synchronization of the constituent parts.

4.2.2 Hardware abstraction layer for the sensor nodes

As CanFestival, the open source implementation of CANopen that was selected for this project, does not strictly require an operating system to run on, if can be differentiated between the hardware nodes on the basis of whether they host an operating system or not. Hardware nodes that are used to interface to the sensors require software of a significantly reduced complexity compared to that of the hardware node that hosts the control algorithm. The sensor nodes need to fulfill these requirements: obtain the data from the respective sensor, process the sensor data into a form that can be passed on to the CANopen stack and enable CanFestival to communicate with the control node over CAN.

The approach of executing CanFestival on the CPU without an underlying operating system is followed for the sensor node software since the STM32F10x Standard Peripherals Library, provided by ST Microelectronics as-is and without licensing limitations, provides the necessary drivers and functionality. Table 4.2 shows which modules of the Standard Peripherals Library were used in the software of each sensor node. Modules common to all sensor node types are CAN for the communication, Flash for the optimal configuration of flash wait states, GPIO for the control of a heartbeat LED, RCC for clock and PLL configuration, TIM for the timebase used by CanFestival and UART for debug messages during development. The magnetic sensor node additionally needs to evaluate the state of six further digital input. The ultrasonic and quadrature encoder sensor nodes use an additional timer unit; the ultrasonic sensor node uses the timer for measurement of distance encoded as duty cycle of a PWM input, while the quadrature encoder node profits from the hardware support for evaluation of a quadrature encoded signal, presented in Figure 3.3. The rotary encoder sensor interface makes use of the external interrupt (EXTI) module as well, to provide a way to reset the timer register containing the pulse counter on each full revolution of the spindle.

Node type / Modules used	Magnetic sensor	Ultrasonic sensor	Quadrature encoder sensor
CAN	X	X	X
EXTI			X
Flash	X	X	X
GPIO	X (six inputs, one output)	X (one output)	X (one output)
RCC	X	X	X
TIM	X (one timer)	X (two timers)	X (two timers)
UART	X	X	X

Table 4.2: Modules of the STM32F10x Standard Peripherals Library used by the sensor nodes

4.2.3 Operating system for the master node

The software that is to run on the control board is significantly more complex in size and functionality, as well as in the fact that hard real-time deadlines influence the requirements on execution of some software modules. On this node bidirectional communication over two UART devices, CAN communication and execution of the elevator control algorithm need to be executed quasi-simultaneously. For this purpose ChibiOS/RT, an open source real-time operating system released under the GNU General Public License, was selected. A detailed description, documentation and examples can be found at the homepage of the ChibiOS/RT project [3], and the source code of the project is available from the project page at Sourceforge [4].

ChibiOS/RT is an operative system built around a preemptive scheduler supporting multiple priority levels, where threads with the same priority level are scheduled using the round robin approach. Although various extensions for dynamic objects are usable, the OS architecture is static and statically allocatable at compile-time. A Hardware Abstraction Layer is provided for numerous ARM-based microcontrollers and evaluation boards. One of the most important reasons for selection of ChibiOS/RT are the support of the STM32F103 microcontroller as well as of the evaluation board "STM32-P103". All of the functionality required to run CanFestival on top of ChibiOS/RT are provided by the hardware abstraction layer and the features supported by the OS kernel. These include an abstracted CAN driver for direct interfacing to CanFestival, as well as the underlying low-level CAN driver. An abstracted view of serial communication is available that can be used for the communication to XINU and the motor control board, that further use the available low-level UART driver. A ports abstraction provides access to the on-board LED for a status notification. The general purpose timer module is used for provide CanFestival with a notion of time. A step function of the CanFestival stack is called with a period of 50 ms in a separate thread and the time it is provided with has a resolution of 10 us.

4.2.4 Flowcharts of the NMT slave nodes

Sensor boards of the system are modeled as slave nodes in the sense of CANopen network management, while the control board is modeled as the master node. The master node, through dedicated NMT CAN messages, controls the network-wide network management (NMT) state diagram illustrated in Figure 4.7. The NMT message uses a network-wide unique CAN ID and a two byte payload with Byte 0 being the command specifier and Byte 1 the Node-ID of the node that is to execute the command. The node-ID 0 is a special broadcast node-ID meaning the command specifier is to be executed by all nodes on the network.

Figure 4.8 illustrates (a) the initialization procedure that is to be executed by sensor boards in the INIT state, and (b) the flowchart to be implemented for the sensor boards in the operational state. Due to the simple structure of sensor nodes, their states referred to in this flowchart can be directly mapped to NMT states of Figure 4.7. The "INIT" state thus maps to the NMT state "Preoperational" and the Operational state to the NMT state "Operational". Sensor nodes do not require a separate safe state but are instructed by the NMT master to switch to the NMT state "Stopped" in which all communication by the sensor nodes ceases.

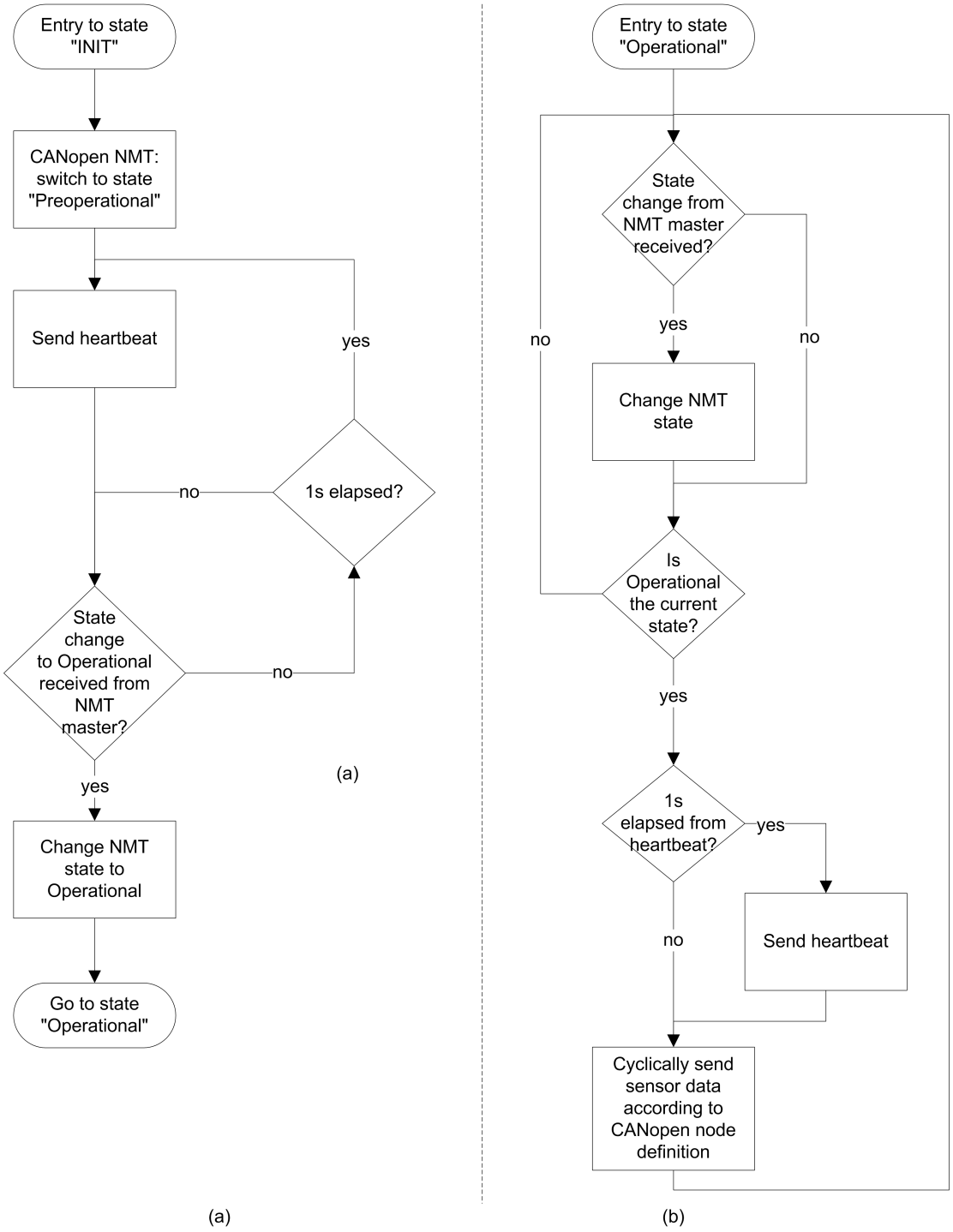

Figure 4.8: Flowcharts of the sensor node functionality in the (a) INIT and the (b) Operational state

4.2.5 Flowcharts of the NMT master node

An abstracted view of the control board functionality in the three main states is presented in a form of flowchart in Figure 4.9. The state INIT is entered as the control board is powered on and finished its local initialization routines. As soon as the master receives one heartbeat message from

each of the sensor nodes the CANopen network can be put into the global state "Preoperational". The control algorithm can then be started and the initial position of the elevator car obtained.

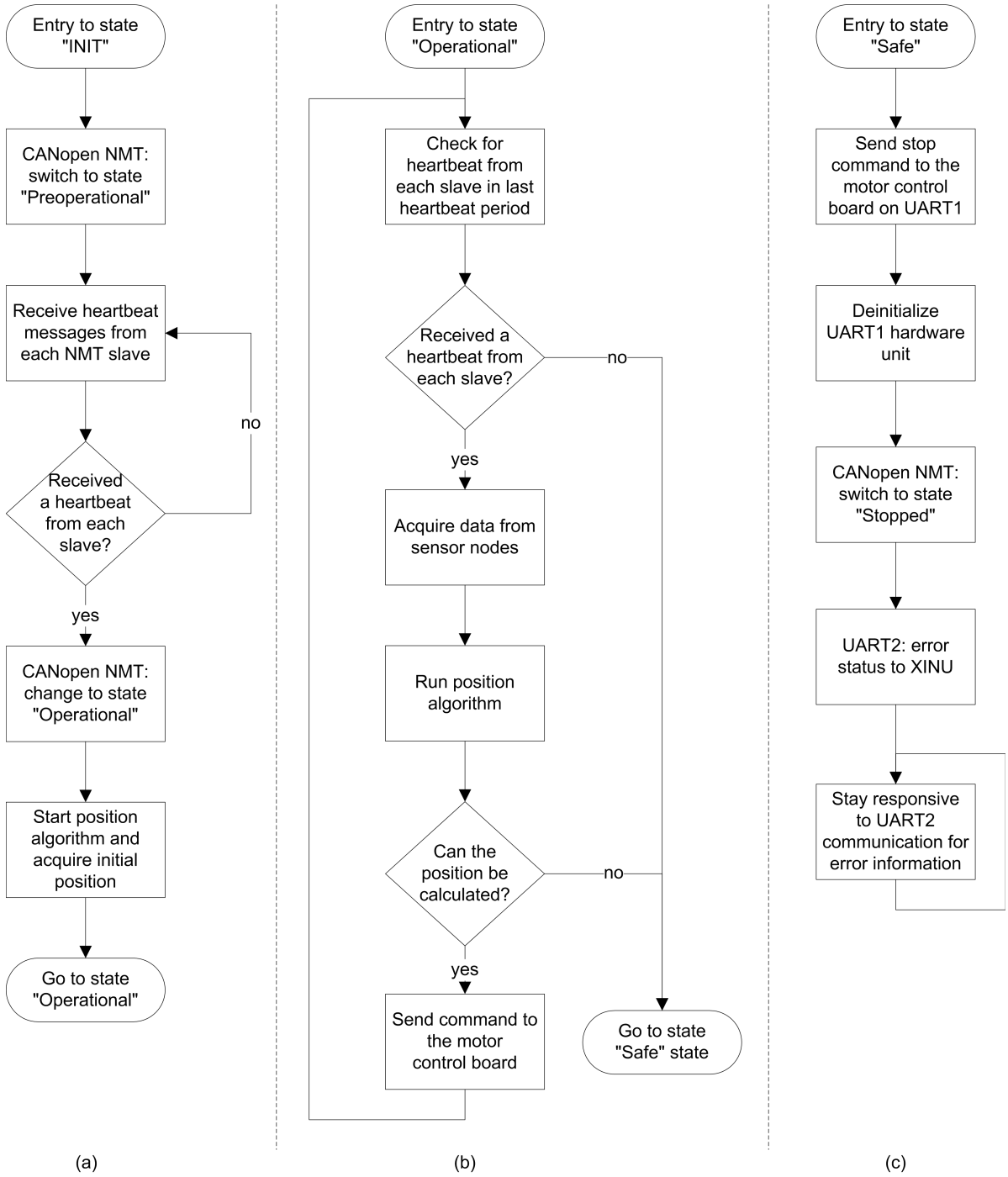

Figure 4.9: Flowchart of the control board functionality in the (a) INIT, (b) Operational and (c) Safe states.

As the control node switches the network into the state "Operational" via the CANopen NMT master, it is checked whether all the sensor nodes communicated with the control board via their respective heartbeat messages. Only if all of the sensor nodes sent their heartbeat messages

within the last heartbeat period the sensor values can be obtained from the Object Dictionary and subsequently used as input for the position check algorithm. Finally, when the elevator car position can be reliably calculated it is used as input to the motor control module.

In the case that the elevator car position can not be reliably calculated, for example due to the failure of one of the sensors, the "Safe" state is entered. In this state the control board instructs the motor control board to stop the motor, ensures that the elevator car remains stationary by disabling the UART communication channel to the control board and puts the CANopen network in the state "Stopped". The control board reports an error to the XINU environment and remains responsive on this UART interface for possible exchange of error and diagnostic information.

A mapping between application states and NMT states similar to that for NMT slave nodes is applicable to the NMT master node and the software running on it. The application state "INIT" maps directly to the NMT state "Preoperational", while the "Operational" state of the application maps to the NMT state "Operational". The application "Safe" state can be mapped onto the NMT state "Stopped", since no further communication relevant for motion control is required in this application state.

5 System implementation

The system implementation can be separated into parts dealing with the hardware and the software. The focus is on the additions to the hardware due to specific interface requirements, bring-up of the evaluation boards, common and specific software for each board type and the configuration and initial start-up of CanFestival and ChibiOS/RT.

5.1 Hardware

Due to the electrical parameters of the microcontroller at the core of the development boards used, interface boards were designed to allow for correct reading of the sensor outputs. Further, the complete hardware of the system is presented along with the bus layout used to connect the hardware parts.

5.1.1 Hardware interface adapters

The STM32F103 evaluation boards are based on the microcontroller STM32P103 produced by STMicroelectronics. The device is to be operated with a voltage supply producing between 2.0 V and 3.6 V [STM12]. However, most of the general-purpose input/output pins are 5 V-tolerant, i.e. they can operate with input voltages of over 5 V.

The magnetic sensors used, as well as the ultrasonic sensor, require a supply voltage of 12 V. This supply voltage is provided using the sensor interface boards presented Figure in 4.1 and Figure 4.3, respectively. Both of these sensors have levels up to the supply voltage as maximum output levels. Since a general-purpose input pin can be used on the magnetic sensor nodes, the maximal incoming signal voltage needs to be lowered from 12 V to 5 V and no further interfacing is required to connect a magnetic sensor to the board. As the pins that are mapped to timer input channels are not tolerant to 5 V inputs, the maximal incoming signal voltage from the ultrasonic sensor needs to be lowered from 12 V to 3.3 V. The ultrasonic sensor provides two additional inputs, a synchronization input and a parametrization input. In the use case where several ultrasonic sensors are used and there exists the possibility of mutual interference between the sensors, synchronization inputs are used to independently trigger the beginning of a measuring cycle for each sensor. The synchronization feature is not required for this project since only one ultrasonic sensor is used and this input is connected to ground, as per the instructions given by the manufacturer. The STM32F103 can detect frequency values of minimally 1.1 kHz. In

order to be able to correctly evaluate the sensor output frequency, this parametrization input is connected to 12 V, as per the manufacturers instruction, causing the sensor to encode the distance information within a signal with a frequency of 1.9 kHZ.

The rotary encoder requires a supply voltage of 5 V. This sensor also uses its supply as the maximum output level and an interface board, presented in Figure 4.2, is used to lower the maximal incoming signal voltage from 5 V to 3.3 V with no further interfacing required.

5.1.2 System hardware

The complete hardware of the elevator control system based on CAN consists of the controller node, three sensor nodes complete with the respective sensor and interface adapter board, the XINU environment and the motor control boards. The controller board is connected via CAN to the sensor nodes, and via an RS-232 connection to the motor control board. An interface for connection to the XINU environment using another RS-232 connection is provided as well. Figure 5.1 shows the system hardware where the scope of the system developed and described within this thesis is bounded by the dashed line.

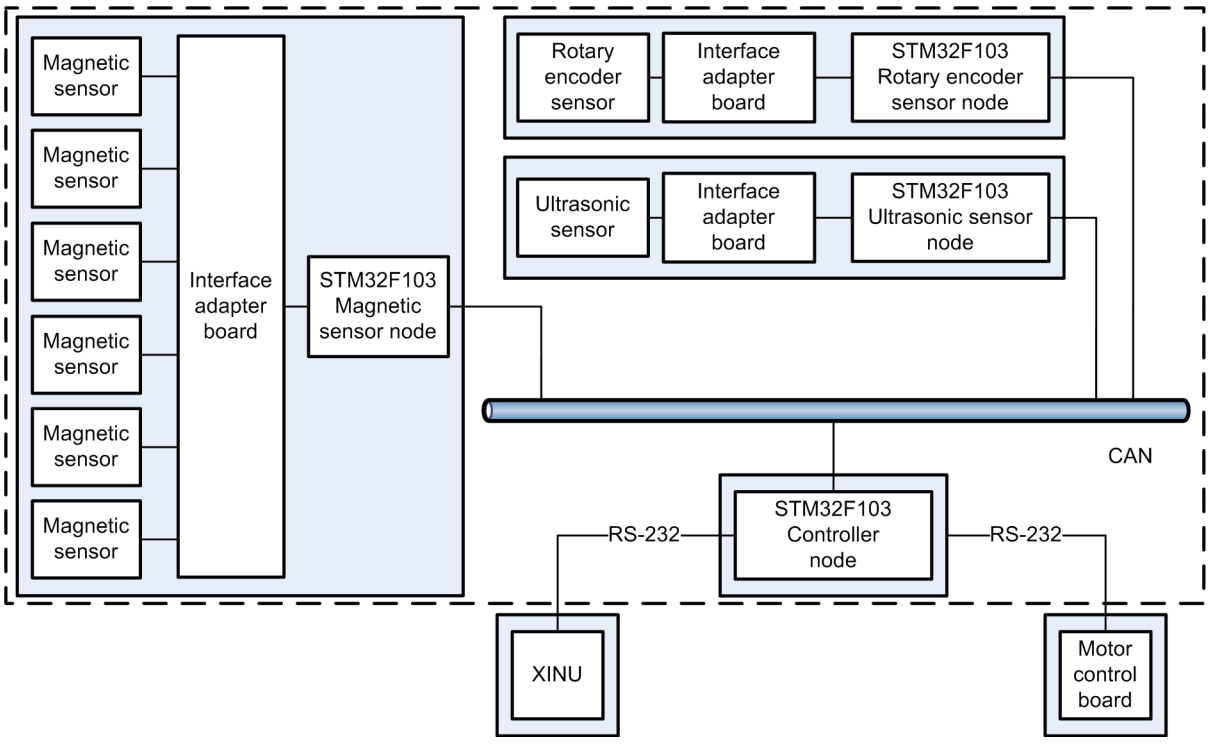

Figure 5.1: View of the system hardware

The CAN bus can be topographically viewed as a single line. However, physically the bus was realized as a two-line twisted pair bus with a termination at each end of the bus. In order to further reduce reflections the bus was not realized as a trunk line with a branch connected to each node, but rather in a hop-by-hop fashion, as shown in Figure 5.2.

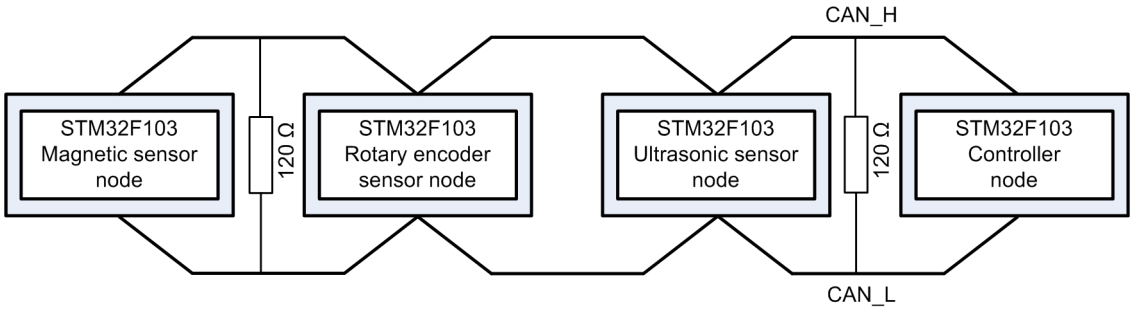

Figure 5.2: Implementation of the CAN bus

5.2 Software

This section describes the measures that were taken during the software implementation. An overview of the development environment and the toolchain is given, as well as the process of porting the CanFestival stack to the STM32F103 microcontroller and integrating the stack into ChibiOS/RT. Additionally, the functionality of slave nodes and that of the master node is described.

5.2.1 Development environment

The toolchain used for development of this project is made up of the following tools:

- Eclipse IDE with C/C++ Development Tooling [12],
- Zylin Embedded C/C++ Development Tooling plugin for Eclipse [11],
- Embedded Systems Register View plugin for Eclipse [14],
- YAGARTO (Yet Another ARM GNU Toolchain) [13],
- Open On-Chip Debugger (OpenOCD) [15].

The OpenOCD Development Suite (ODS) put together by Olimex was used as the basis for the development environment. It includes the Open On-Chip Debugger with profiles for use with the ARM programmer and debugger tool used with the evaluation boards. The compiler, linker and build support tools of the YAGARTO toolchain, integrated into the Eclipse IDE by their installer, were used for the build process. OpenOCD was used from withing Eclipse to connect to the on-chip debugger module of the STM32F103, to program the microcontroller and to debug the software. Separate Eclipse projects for each of the sensor nodes are available, with makefiles tailored to the needs of the node. The controller node software was integrated into the Eclipse project provided by the installation of ChibiOS/RT.

A unit test framework was set up for use with Eclipse to enable automated unit tests of the developed software. An open source unit test framework for C called 'Unity', available from [16], was used for this purpose and the tests were executed automatically at each file save operation.

5.2.2 Porting of CanFestival to STM32F103

The CanFestival implementation is a platform-independent implementation of the CANopen stack with the source code available from [7]. CanFestival is grouped into a static part, implementing the functionality of each supported service, and a configuration part that represents data upon which the static functionality is executed. Examples of the stack ported to various platforms are included in the repository, including Linux and Windows-based PCs with external CAN interfaces as well as examples of several embedded targets.

As the source code is written in a platform-independent way, the process of porting of CanFestival was a matter of recognizing and correctly implementing the wrappers that adapt the target-specific functionality to the interface used by CanFestival. For embedded targets the porting process requires the use of a timer that needs to be able to generate and handle an interrupt on a certain compare value. CanFestival includes a micro-scheduler that manages an alarm table and dispatches calls to stack functionality through callbacks at the correct time. This dispatcher can to be called at regular intervals from a periodic task in the case that an operating system is used, but for better resolution of triggered events it should be possible to specify the next trigger point.

Further, CanFestival requires access to the CAN interface of the microcontroller in the form of functions that initiate the transmission of a CAN message from, and pass an incoming message to the CanFestival stack . As CanFestival abstracts from the CAN controller itself, indeed it does not differentiate whether an MCU-internal or external CAN controller is used, it is the responsibility of the system integrator to make properly configure the CAN controller.

Access to the Object Dictionary is implemented using access functions, although direct access to the arrays storing the mapped variables is available for performance purposes.

5.2.3 Integration of CanFestival and ChibiOS/RT for the controller node

As described in section 5.2.2, the major task when porting and integrating CanFestival into a new system is the provision of interface functions regarding the timing and access to CAN messages. ChibiOS/RT can be used to provide CanFestival with a notion of time through use of a general purpose timer module (GPT). An general purpose timer presents an abstracted view of a timer unit that can be used for triggering of a callback function. Additionally, a function was provided to enable CanFestival to set an alarm by influencing the next scheduled expiration point of the timer.

Wrapper functions were required to adapt the calls of generic transmission and reception functions of CanFestival onto the ChibiOS/RT-specific CAN transmit and receive functions. Functions provided by ChibiOS/RT were used for configuration of the CAN controller itself and the configuration values for the timing were synchronized to those used by the sensor node. Since the software running on the controller node needs access to the data sent by all of the sensor nodes its acceptance filter for CAN IDs needs to be set wider that those of the sensor nodes. Apart from the process data, i.e. sensor data values, the controller node also receives and processes the heartbeat messages sent by the sensors nodes.

5.2.4 Functionality of slave nodes

All of the sensor nodes were designed using the same structure. Since no further functionality was required besides the acquisition of their respective sensor values and the CanFestival stack, the sensor node software consists of an initialization procedure and a looped portion in which the sensor value is read and forwarded to CanFestival. The initialization procedure for each sensor node is slightly different as each sensor node requires a unique node-ID and a sensor-specific method of peripheral initialization. Configuration of the CAN controller is shared among sensor nodes, as are the time-handling configuration and the corresponding interrupt service routine.

The slave node connected to magnetic sensors needs a simple configuration since it connects to six sensors that can be abstracted as digital inputs. The software on this node reads the state of each of the digital inputs, forwards the aggregated values to the CanFestival stack running on the node. The stack packs the values into a transmission process data object (PDO) and sends the PDO to the controller node according to the configuration.

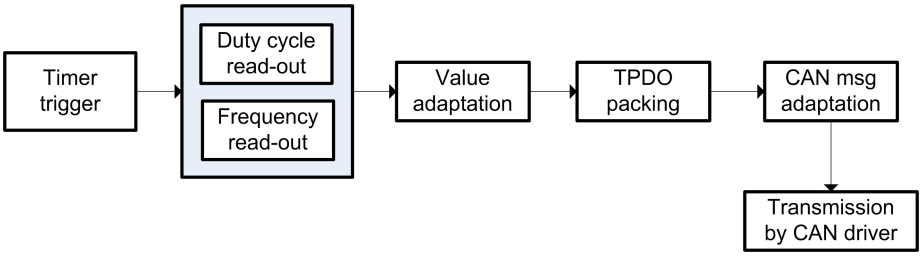

Figure 5.3: Block diagram of sensor nodes software on the example of the ultrasonic sensor node

The other two slave nodes that interface with the ultrasonic sensor and the rotary encoder have a slightly extended form shown in Figure 5.3. An additional timer unit that provides hardware-support for the evaluation of the frequency and duty cycle of a pulse-width modulated signal requires a separate configuration. The distance is conveyed by the sensor encoded in the duty cycle of the signal, and relates to the physical distance of the object from the sensor as shown by equation (5.1), where the sensing range is specific to the ultrasonic sensor used and equals 1000 mm.

$$distance_{object} = sensing\_range * pulse\_length * frequency \tag{5.1}$$

An additional timer unit of the microcontroller is used on the rotary encoder sensor node to provide hardware support for the acquisition of the spindle rotation information. The pulses on tracks A and B are output by the sensor with a phase shift of 90° in the case of clockwise shaft rotation, and with a phase shift of -90° for anticlockwise shaft rotation. The phase is detected and stored in a separate direction register of the peripheral. In encoder mode the timer acts as a counter of pulses detected on the two tracks A and B, counting up for a clockwise and down for an anticlockwise rotation direction. This is the principle of operation of a relative rotary encoder as the sensor does not give any information regarding current position of the shaft within regard to a certain fixed angle. Thus the controller node includes a position resolution algorithm that initializes the internal representation of the shaft position at start-up of the controller node.

5.3 Supporting tools

During the project need for tools arose that would make the development, configuration and test processes simpler and more comfortable. Some freeware and tools in the public domain were found to be very useful and fit for their purpose, while others need to be written specifically for use in this project. This section describes the most important and non-obvious tools used to support the development process.

5.3.1 Objdictedit

The Object Dictionary Editor is a GUI-based configuration support and code generator tool for CanFestival written in Python. It is used to make configuration of the object dictionary of a CANopen node as user-friendly as possible and to provide a uniform implementation of the object dictionary concept. The tool itself is a part of the CanFestival repository and can be obtained in a version synchronized to the source from [7].

Objdictedit provides support for profiles dealing with the application layer and communication as outlined in [CAN02], as well as with several devices such as generic input/output modules described in [CAN08]. Support for further device profiles can be extended through use of the profile mapping mechanism that allows Objdictgen to function in a profile-independent way through storage of profile-specific data in profile mapping files in a specific folder of Objdictedit. This functionality allows simple extensibility and support for new or changes profiles.

5.3.2 CANHacker

CANHacker is a shareware monitor and tracer tool for the PC that can be used to communicate to several types of CAN interface hardware. The tool is hosted and supported through a forum available at [9]. The software tools can be downloaded using the link [8] which demands a registration to the aforementioned forum.

The tool can be used to display the contents of all received CAN messages in the tracer mode, or display the contents of the most recent CAN message with a certain CAN-ID in the monitor mode. Further useful functionality is the support for comments with the help of which each CAN-ID can be described in order to make analysis of bus traffic easier.

As CANHacker is fundamentally an interface and visualization tool it requires a hardware counterpart that features a CAN transceiver, an interface to the PC running CANHacker and a microcontroller that forwards the CAN traffic to this interface. A spare OLIMEX STM32F103 evaluation board was used as the hardware counterpart since it features all of the hardware required for this purpose and significant portions of the software relating to the initialization of CAN could be shared among this software and that used for the slave boards.

CANHacker supports several communication protocols used by hardware interfaces. One of these, described at [6], is the LAWICEL protocol for communication interfaces between a CAN-bus and USB. The hardware implementation actually uses a virtual serial port that needs to be selected within CANHacker. Apart from basic reception, CANHacker further supports functionalities such as simulation of CAN messages and transmission request with standard or extended ID fields, use of custom bit-rates and acceptance filters for incoming messages.

Firmware for the hardware interface was written, implementing the reception and timestamp capabilities specified in [6], to be run on a spare STM32F103 board and forward received CAN data over a RS232-to-USB adapter to a PC running CANHacker in order to display the data. The received data can be presented by CANHacker either using the monitor or the tracer mode. Only the relevant portions were implemented, namely the setup of a "Listen Only" mode on a 125 kbit/s CAN-bus for reception of messages with a timestamp resolution of 1 ms , as the protocol functionality is quite extensive and was not required in its full breadth.

6 Results

A prototype of an elevator control system was developed according to the design goals and artifacts described in Chapter 4. This chapter presents an analysis of the achieved design goals and the results obtained during operational testing.

6.1 Fulfillment of requirements

The ten recognized hardware and software requirements presented in Figure 3.4 form the basis for an analysis on the fulfillment of requirements.

Requirements on hardware can be directly mapped to hardware design artifacts presented in Chapter 4 on System design. Their fulfillment can be followed to the system hardware that was shown in Figure 5.1. The design artifact regarding the use of a particular evaluation board was trivially fulfilled, while those artifacts related to the ability to interface with the necessary sensors are fulfilled through the use of interface adapter boards for each of the three sensor types. Support for CAN as the communication system was also trivially solved by the use of the "STM32-P103" evaluation board that features an on-board CAN transceiver and a selectable termination resistor. The design artifact on RS-232 connectivity with two peers was fulfilled by the use of the on-board RS-232 interface for communication with the motor control board, and an additional off-the-shelf UART-to-USB adapter cable.

The software requirement on the ability to accept instructions from the XINU command over the RS-232 interface can be directly mapped to a task configured to start upon reception of data through the serial connection. However, the requirements on the precision of position data and the time to enter the safe state upon detecting an invalid position could not have been met in the form that they were assumed at the start of the project. Due to the nature of the sensors used, a comparison of sensor values and the subsequent calculation of validity of the position value aggregated from the three sensor types can only be made when the elevator car is at one of the floor positions, or is nearing one such position. This is a fundamental restriction on the system which could only be mitigated by exchanging the magnetic sensors with another sensor type, one which would function over a denser range.

A further issue was observed with the rotary encoder sensor. The current elevator model introduces a significant amount of slip between the rotating electric motor and the spindle it drives. The slip is highly dependent on the load related to the direction of movement of the elevator car, as well as by the friction between the rails along which the car is being guided and the brackets

on the car. The connection of the rotary encoder sensor to the rotating spindle is shown in Figure 6.1.

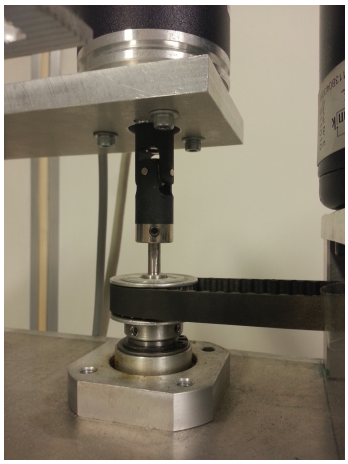

Figure 6.1: Rotary encoder sensor

Due to the low effective resolution of the rotary encoder sensor and the fundamental limitation of continuity with the magnetic sensor inputs from these two sensors are used only in the module that calculates the confidence in the sensor values. This calculation is triggered when values from all three sensors can be trusted, i.e. when the elevator car is static at one of the floor positions. The ultrasonic sensor, as the one with the best combination of accuracy and continuity is used for movement control between two floor positions.

6.2 CAN bus load

In the implementation described in the previous chapter a number of CAN messages were sent by each network node. Table 6.1 gives a complete view of the messages transmitted over the CAN bus and their most important parameters.

Message ID	Message length in bytes	Period	Description
0x001	2	event-triggered at start-up and shutdown	NMT state change
0x182	1	100 ms	magnetic sensor node TPDO1
0x183	3	20 ms	rotary encoder sensor node TPDO1
0x184	2	50 ms	ultrasonic sensor node TPDO1
0x701	1	1000 ms	heartbeat - controller node
0x702	1	1000 ms	heartbeat - magnetic sensor node
0x703	1	1000 ms	heartbeat - rotary encoder sensor node
0x704	1	1000 ms	heartbeat - ultrasonic sensor node

Table 6.1: CAN messages on the bus

The total load of the CAN bus taking into account the messages presented in Table 6.1 is calculated employing a method analog to the one used in section 2.5. A static evaluation is appropriate in this case although there is one event-triggered message in the system, the one that the master network management node uses to change the global state of the network. The state change is triggered by the master only at system start-up and shutdown, as well as upon entering the safe state which does not significantly add to the bus load.

Within a reference period of one second, which is the longest message period on the bus, 8386 bits are transmitted on the CAN. This worst-case calculation takes into account the message ID and control information, the payload and the CRC bits of each message. The calculated bus load amounts to approximately 6.7088% and can be considered low and thus acceptable. Although the approach taken in this project was different to that of using minimum bit message lengths presented in [HKT09], the overall bus load achieved indicates that the system can be extended to over 100 floors before reaching a bus load of 50%.

7 Conclusion

The challenges facing current-day elevator systems are far from simple and elevator control systems can not be viewed as a completed development. Be it in aspects of passenger safety, dependability, availability, comfort of use and travel or security against attacks, improvements are continually being made. This diploma thesis covered some possibilities that could be employed to raise the level of dependability of an elevator control system. The approach taken was that of using diversity in hardware and software in order to mitigate common-mode failures and design faults, instead of mere multiplication of a system or its parts.

An implementation phase was preceded by an analysis of the input requirements into such requirements that could be traced through design decisions onto the actual implementation artifacts.

A distributed control system was developed on the basis of the design, consisting of a hardware platform with its parts interconnected using a common bus and a software architecture that uses a communication system on top of the bus to enable its modules to exchange information. Three communication systems currently in use in the automotive field were compared using several criteria and investigated for use in an application for building automation. Controller Area Network was chosen to provide the lower two layers as described by the ISO/OSI model. A communication stack suitable for a CAN-based application was chosen and an open source implementation was ported to the microcontroller used in the hardware platform. An important goal during the implementation and integration of software modules was to enable the highest level of reusability of modules and configuration items. To this end the software being executed on each of the three sensor node has an equal structure, with only the parts dependent on the sensor interface parts being variant.

The control system was integrated into an elevator model at the Institute of Computer Technology of the Vienna University of Technology. The complete elevator model consists of two shafts, each of which includes an elevator car, six floor positions, an electric motor to drive the car along a rotating spindle and a series of sensors that can be used to evaluate the position of the car.

The control system is set in the context of a larger system centered around the project eXellent Interface for Non-hapic Use, XINU for short, created with the aim to research interfaces for non-haptic control that would promote the concept of inclusion. The interface to XINU is a serial one, using which the XINU environment can send the next floor position for the elevator car to move into. All preparations necessary for the use of this interface with an implementation of XINU have been made and the interface was used during operation and testing of the system to provide set-points for the elevator to move to.

The developed system, as integrated into the elevator model, was put into operation and tested, yielding input for improvements to the sensor concept and the mechanics of the elevator model. It was shown that the system can function properly even in a state of degraded quality of sensor input and can detect failures of each of the sensors, subsequently entering the safe state, stopping the elevator car and disabling further control of the electric motor. A static analysis of the load on the CAN bus shows a very low load at the specified bit-rate and demonstrates the property of the system using the same or a similar concept for acquisition of sensor values and motor control.

Several key points of improvement were identified during this project and can be grouped into the following categories.

- Revision of the sensor concept: Due to the sensitivity of some sensors on the mechanics of the system a revision of the sensor concept is necessary. In particular, an extension or replacement for the rotary encoder sensor is to be investigated that would decouple the measurement of the car position from the rotating spindle as this is the major source of error. The feasibility of accelerator sensors that can be placed into the elevator car should be investigated.
- The resolution controller as shown in Figure 1.3 that would grant access to the motor controller board to one of the two diversitary application controller boards was not within scope of this work. A concept for the resolution controller is necessary in order to enable the full functionality of the elevator control system based on diversity envisaged. Special care is to be taken during the concept development since the resolution controller would present a single-point-of-failure of the system and would thus need to be developed to be as simple as possible, or to be itself diversitary.
- In order to support the use case of selecting the floor to move to from the elevator car a model of the elevator car is required that would optimally include the necessary input buttons, an indicator of the current floor position and a working or simulated car door movement. Support in terms of an additional sensor node and the corresponding software and configuration changes to the master node will thus be required.
- With the addition of a modeled elevator car and taking extendability of the system in mind, the feasibility of an implementation according to the CAN in Automation specification CiA417 [CAN10b] should be reconsidered.

Literature

[Bos91] Bosch R.: *CAN specification, version 2.0 Part A*. 1991

[Bos95] Bosch R.: *CAN specification, version 2.0 Part B*. 1995

[BSCF98] Barbosa, M. B. M. ; da Silva Carvalho, A. ; Farsi, M.: A CANopen I/O Module: Simple and Efficient System Integration, 1998

[CAN02] CAN in Automation: *CiA 301 CANopen Application layer and communication profile*. 2002

[CAN07] CAN in Automation: *CiA Draft Recommendation 303 CANopen Additional Specification Part 1: Cabling and connector pins assignment*. 2007

[CAN08] CAN in Automation: *CiA 401 Device profile for generic I/O modules*. 2008

[CAN10a] CAN in Automation: *CiA 102 CAN Physical layer for industrial applications*. 2010

[CAN10b] CAN in Automation: *CiA 417 Application profile for lift control systems*. 2010

[CLXC06] Chen, Z. ; Luo, F. ; Xu, Y. ; Cao, J.: Design and Implementation of Modern Elevator Group Control System. In: *Proceedings of the Fifth International Conference on Machine Learning and Cybernetics*, 2006

[Eur07] European Committee for Standardization: *ISO 13849-1 Safety of machinery - Safety-related parts of control systems - Part 1: General principles for design*. 2007

[Fle05] FlexRay Consortium: *FlexRay Protocol Specification Version 2.1 Revision A*. 2005

[Fle06a] FlexRay Consortium: *FlexRay Electrical Physical Layer Application Notes Version 2.1 Revision B*. 2006

[Fle06b] FlexRay Consortium: *FlexRay Electrical Physical Layer Specification Version 2.1 Revision B*. 2006

[HKT09] Huseinbegovic, S. ; Kreso, S. ; Tanovic, O.: Design and Implementation of the CAN Based Elevator Control System. In: *XXII International Symposium on Information, Communication and Automation Technologies*, 2009

[ISO03] ISO - International Organization for Standardization: *ISO 11898-2, Road Vehicles: Controller Area Network (CAN) Part 2: High-speed medium access unit*. 2003

[KCS00] Kim, T.S. ; Cho, H. ; Sung, D. K.: Moving Elevator-Cell System in Indoor Buildings. In: *IEEE Transactions on vehicular technology*, 2000

[LHR+09] Livint, G. ; Horga, V. ; Ratoi, M. ; Albu, M. ; Chiriac, G.: Implementing the CANopen Protocol for the Distributed Control of a Hybrid Electric Vehicle, 2009

[LIN10] LIN Consortium: *LIN Specification Package Revision 2.2*. 2010

[Mai11] Maier, B.: *Fehlertolerante Aufzugsteuerung*, Technische Universitt Wien, Diplomarbeit, 2011

[MSM99] Mitra, S. ; Saxena, N. ; McCluskey, E.: A Design Diversity Metric and Reliability Analysis for Redundant Systems. In: *Test Conference, 1999. Proceedings.*

International, 1999

[PNH04] Pereira, C. E. ; Netto, J. ; Hsemann, R.: A CANopen Profile for Multi-Tap Auto-Transformers, 2004

[RBF99] Ratcliff, K. ; Barbosaand, M. ; Farsi., M.: An overview of controller area network. In: *Computing Control Engineering Journal*, 1999

[RPZH10] Roesener, C. ; Perner, A. ; Zerawa, S. ; Hutter, S.: Interface for Non-haptic Control in Automation. In: *8th IEEE International Conference on Industrial Informatics*, 2010

[STM10] STMicroeletronics: *RM0008 Reference manual STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM-based 32-bit MCUs*. 11. April 2010

[STM12] STMicroeletronics: *STM32F103x8 STM32F103xB Datasheet*. 14. December 2012

[WXC+08] Wengang, J. ; Xiangdong, J. ; Canfeng, Z. ; Tiexiang, L. ; Lixin, Z.: Research on the Control System for Automatic Pipeline Welding Based on CANopen, 2008

[ZRP11] Zerawa, S. ; Roesener, C. ; Perner, A.: Non-haptic Interaction System. In: *IEEE International Symposium On Industrial Electronics (ISIE)*, 2011

Internet References

[1] Encyclopdia Britannica Online. *Encyclopdia Britannica Online, s. v. "Elisha Graves Otis".* Accessed March 2013. www.britannica.com/EBchecked/topic/434691/Elisha-Graves-Otis.

[2] Institute of Computer Technology. *Homepage.* Accessed March 2013. www.ict.tuwien.ac.at.

[3] ChibiOS/RT *Project Homepage.* Accessed March 2013. http://www.chibios.org.

[4] Sourceforge *ChibiOS/RT free embedded RTOS.* Accessed March 2013. https://sourceforge.net/projects/chibios.

[5] ST Microelectronics *STM32F10x Standard Peripherals Library V3.5.0.* Accessed March 2013. http://www.st.com/web/en/catalog/tools/PF257890.

[6] LAWICEL AB *CANUSB Manual* Accessed May 2013. http://www.canusb.com/docs/canusb_manual.pdf.

[7] CANFestival *CANFestival code repository* Accessed May 2013. http://www.canfestival.org/code.

[8] CANhack *CANHacker V2.00.02* Accessed May 2013. http://www.canhack.de/download.php?id=2.

[9] CANhack *CANhack support forum* Accessed May 2013. http://www.canhack.de/.

[10] LAWICEL AB *CAN Tools from LAWICEL AB* Accessed May 2013. http://www.canusb.com.

[11] Zylin Consulting *Zylin Embedded C/C++ Development Tooling* Accessed November 2012. http://opensource.zylin.com.

[12] The Eclipse Foundation *Eclipse C/C++ Development Tooling* Accessed November 2012. http://www.eclipse.org/cdt/.

[13] YAGARTO website *Yet Another GNU ARM Toolchain* Accessed November 2012. http://www.yagarto.de/.

[14] EmbSysRegView project website *Embedded Systems Register View plugin for Eclipse* Accessed February 2013. http://embsysregview.sourceforge.net/.

[15] OpenOCD website *Open On-Chip Debugger* Accessed November 2012. http://openocd.sourceforge.net/.

[16] Github *Unity test framework repository* Accessed December 2012. https://github.com/ThrowTheSwitch/Unity.